MASTER'S DEGREE THESIS

Master of Science in Advanced Mathematics and Mathematical Engineering

Title: Traffic modelling for Big Data backed telecom cloud

Author: Anna Via Baraldés

Advisor: Marc Ruiz Ramírez

Co-advisor: Luís Domingo Velasco Esteban

Department: Computer Architecture

Academic year: 2015-2016

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat de Matemàtiques i Estadística

Acknowledgements

I would like to specially thank my advisors, Marc Ruiz and Luis Velasco, for all their help, suggestions and time spent in the project, and also for deciding to grant me with the GCO-AC scholarship to join the GCO during this academic year. I would like to express my gratitude to the other people in the GCO as well, for kindly taking me in, helping me by explaining doubts and giving me valuable tips and advice.

Finally, I would like to thank my family and friends for their support during the Bachelor and the Master.

Index

List of Figures

List of Tables

Chapter 1.

Introduction

1.1 Motivation and objectives

The introduction of new services and applications requiring large and dynamic bitrate connectivity can cause changes in the direction of the traffic in metro and even core network segments along the day. For instance, Live-TV and Video on Demand (VoD) distribution is in the portfolio of many telecom operators aiming at entering into competition with on-line, over-the-top broadcasters, such as Netflix [Ru16-1]. To facilitate the introduction of new types of service, the introduction of cloud infrastructure in the telecom operator's network (also known as the *telecom cloud* [Ve15]) is a challenging but promising task in the constant evolution of telecom infrastructures.

With the incremental amount of applications running over the telecom cloud architecture it is becoming of paramount importance being able to run simulations aiming at evaluating the performance of such applications. To that end, one of the key elements in the simulation is how to generate network traffic. Several traffic generators have been developed so far, mainly focused on generating representative IP traffic for packet-based networks or connection arrival based on the Poisson distribution for circuit-switched networks [Ca12]. However, changes on the type of traffic nowadays present in the networks arises the need of new traffic models. Specifically, an intermediate traffic generation is needed in between packet generation and connection arrival modelling to reproduce continuous traffic typically observed in cloud-based applications.

The lack of traffic models for new services affects also telecom cloud optimization. Being able of predicting accurate future traffic matrices for periodical network planning [Ve16-1] or rapidly detecting unexpected traffic behavior in the event of an external cause such an attack or a disaster [Na16] are only two examples of application of traffic models in the context of network planning, operation, and reconfiguration. The success of such traffic modelling applications is strongly

related to the adoption of Big Data- backed telecom cloud infrastructures. In this regard, a big data network manager architecture to support network optimization, based on traffic prediction from applying data analytics on the monitored traffic data, has been recently proposed to fully accomplish with a decision making process based on the *observe-analyze-act* loop [Gi16].

The objective of this project is to provide traffic models based on new services characteristics (i.e., continuous, time-variant, heterogeneous traffic). Specifically, we focus on modelling the traffic between origin-destination node pairs (also known as OD pairs) in a telecom network. Two use cases are distinguished: *i*) traffic generation in the context of simulation, and *ii*) traffic modelling for prediction in the context of big-data backed telecom cloud systems. To this aim, several machine learning and statistical models and technics are studied and combined in order to find the best approach for every use case. To evaluate the applicability of selected models, we integrate them in an OMNeT++ network simulator whose implementation follows the Big Data analytics architecture presented in [Gi16].

In order to achieve this objective, strong mathematical background for designing and understanding the models is essential, as well as knowledge on simulation processes is necessary for a successful integration and validation of the models into the aforementioned simulator.

1.2 Report organization

The remaining of this document is organized as follows. Chapter 2 provides a background of the topics that are the base of this project. It starts with an introduction to optical transport networks and follows an introduction to data analysis, to later explain how the network can benefit from it through the Big data analytics architecture on which the Big data telecom cloud is based. It finishes with an introduction to the OMNeT++ simulator, later used to simulate the network traffic.

In Chapter 3 the modelling methodology that is used throughout the project is introduced. It contains some statistical and machine learning techniques used to preprocess the data sets and produce models, and model selection and validation methods. It also contains some self-implemented algorithms that are later used.

Chapter 4 focuses on the specific implementation in the OMNeT++ simulator of the generation of traffic, estimation of models and model validation. In Chapter 5 a real data set is analyzed and several statistical models and machine learning models are produced in order to predict future traffic. The performance of all this models are compared, and from this we are able to decide which model fits the best depending on the purpose or other conditions of the model.

Chapter 6 presents the obtained simulation results, comparing the performance of different predictive models to different profiles of traffic. Also, the recomputation of models, in order to always have valid models predicting traffic, is discussed.

Finally, Chapter 7 concludes the report and highlights the main contributions arisen from this work.

Chapter 2.

Background

In this section the principal topics and concepts that are used throughout the project are explained. Firstly, the cloud-ready optical transport networks are introduced, and an explanation of the Big Data Backed telecom clouds through its Big Data analytics architecture follows, as it is the base architecture on which we work. Next, the OMNet++ simulator is introduced, as it is the framework on which we rely to implements and test the designed processes and models.

Finally, data analysis is introduced in order to understand its goals and problems and to give us a methodology to perform analysis on data sets and to obtain the models later on in the project. The relation between Statistics and Machine learning follows to understand why we chose to combine both fields for the production of models in the project.

2.1 Cloud-ready optical transport networks

An optical network can be defined as a graph with its representative equipment based on a certain optical technology. In general, it is represented by an undirected graph where the edges are fiber optic links and the vertices are optical nodes, named as Optical Cross Connects (OXC), capable of establishing and tearing-down optical connections. The optical technology uses a range of frequencies of the total Optical Spectrum (OS), measured in Gigahertz (GHz). The capacity of an optical link depends on the OS width and other factors like the spectral efficiency of the established connections.

On top of described optical layer, large packet nodes (e.g., IP routers or Ethernet switches) collocated with some OXCs serve as end points of network traffic, as well as to support intermediate transit routing/switching. Thus, a OD traffic flow represents an amount of bitrate transported between a source packet node (s) and a

termination packet node (*t*), usually expressed in Megabits per second (Mb/s) or Gigabits per second (Gb/s).

To support such OD traffic, optical connections in the optical layer are established; these optical connections are called as *lightpaths* since they allow the data transmission as a light wave. From the abstracted view of the packet layer, a lightpath is considered as a virtual link directly connecting two packet nodes. Thus, a virtual topology is created and used to convey OD traffic between source and destination nodes.

The simplified network architecture required to understand the contents of this project is presented in Figure 2-1, where an optical transport network containing a set of packet nodes are interconnected by means of virtual links of the packet layer. Each virtual link is supported by one or more optical connections in the optical layer and OD traffic is served through such capacity. For the sake of simplicity, details on network connectivity are not depicted in the figure.

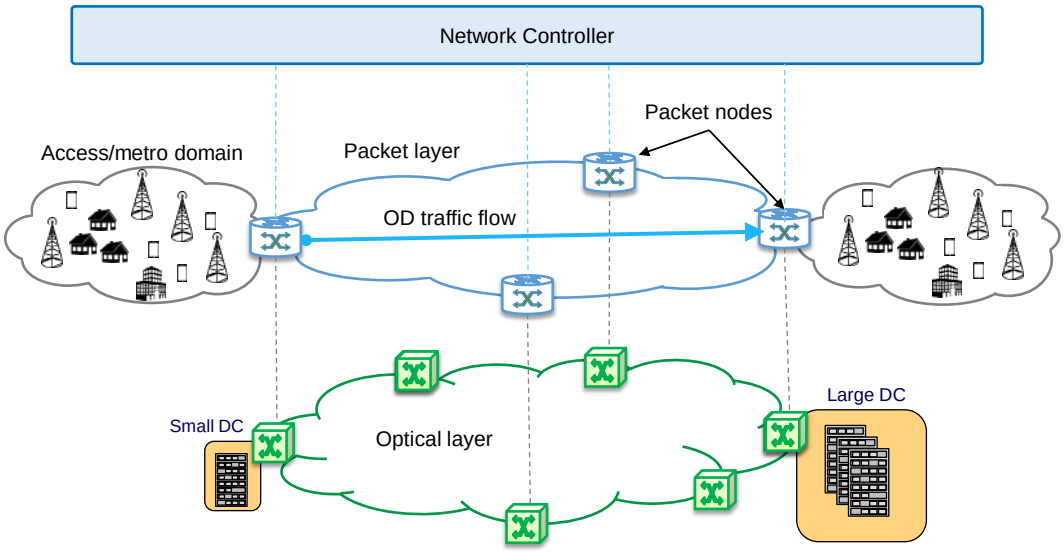

Figure 2-1: Considered network

The network in the example interconnects different access and metropolitan areas where users are. All traffic generated in one area targeting other area in the operator's domain or another network (e.g., the Internet) is aggregated in the source packet node and sent towards the destination node becoming an OD traffic flow.

As introduced in Section 1, the portfolio of network operators is being extended with new types of service requiring not only data transport but also computing (e.g., video trans-rating required for distributing Live-TV services). Thus, we assume that a number of large and small datacenters (DCs) are owned by the operator and interconnected among them and with the users through the transport network. Note that actions such as distributed computing or database

synchronization entails traffic among datacenters that obviously impacts on OD traffic. For instance, in the example above, the depicted OD flow can include at the same time: *i*) traffic between end users of both areas, *ii*) traffic between a source area and a large DC, and *iii*) traffic from small to large DCs.

The considered cloud-ready transport network requires dynamic control of both network and computing resources. In fact, orchestration between cloud and interconnection network is required to coordinate resources in both strata in a coherent manner, which is done by means of an intelligent network controller. Although no specific technology is strictly assumed for this control, the Application-Based Network Operations (ABNO) architecture proposed by the Internet Engineering Task Force (IETF) can be used as a centralized entity in charge of controlling the network in response to requests from the applications and services [RFC7491].

2.2 Basic concepts on Data Analysis

Dealing with big amounts of data arises many problems: the need of manipulation techniques to deal with the size, limited computer memory, increasing running time complexity of the programs, and need of real time predictions (production of data is now non-static, new information can be produced in very small periods of time), [Ha09].

The goals when analyzing data are to make accurate predictions of response variables given future input variables and to figure out the association of the response variables.

There are many important things to have in mind when designing models, [Bre01]:

- The simpler, the better: there exists the claim that the simpler a model is, the better (it is much more understandable and less susceptible to overfitting). Simplicity may imply less accuracy, so one has to find a balance between simplicity and accuracy of the model.
- The curse of dimensionality: from the statistics point of view, dimensionality is a problem, and the number of dimensions are reduced with methods such as Principal Component Analysis to get more significant variables. Other work shows, though, that high dimension can be a good thing, for example Support Vector Machines combine variables to produce an even higher dimensional space.
- Multiplicity of good models: the number of possible good models for the same database can be quite big (different models can have the same minimal error). This also means (looking at the relative weights each variable ends up having at each model), that the influence of each predictor variable to the response variable might also changes, which is not at all intuitive.

Since there are usually unmeasured variables producing noise to the response variable, prediction will not be perfect. One can measure the errors of prediction and make a comparison to choose among the possible models. But if a model has too many parameters, overfitting may occur, and because of this, a penalty related to the number of parameters of the model should be applied to each model.

One of the main goals of the fields of machine learning and statistics is to produce methods and techniques to perform prediction. Both fields have been and are being developed to extract as much knowledge and make as accurate predictions as possible from the huge amounts of information nowadays available

Statistics assume data has been generated by a given stochastic model. The goal is to end up with a picture of how the predictor variables affect the response variables. Validation of a model is done by assessing goodness of fit on residuals, and hypothesis testing. Examples are linear regression, logistic regression, generalized linear models and so on. It typically deals with very particular datasets: clean, small, static, sampled in an i.i.d. manner, numeric and collected with a particular set of questions in mind.

Machine learning's goal is to build computer systems that can adapt and learn from their experience, [Di99]. Some tasks can be hard to define except via examples. Studying how to reproduce the human learning system has brought a new way to understand data. Examples are decision trees, neural networks, support vector machines and so on.

Some people see Statistics and Machine learning as two totally different approaches to data analysis, but actually they are not completely disjoint. Some scientists see machine learning just as a different point of view of the statistical methods, with new notations and new fitting methods, but basically the same, though machine learning can be more intuitive.

At the end, the best model could be a machine learning model, or a statistical model or a combination of both, but one should consider as many tools as possible in order to arrive to the best possible model.

2.3 Big data analytics architecture

Traffic monitoring is an essential task for network operators since it allows evaluating network performance. To perform control upon the network, data analytics can be carried to the observed traffic. To that end, data is recollected and appropriately stored, preprocessed, and modelled by predictive models that indicate the future evolution of the traffic. More specifically, the overall architecture is shown in Figure 2-2.

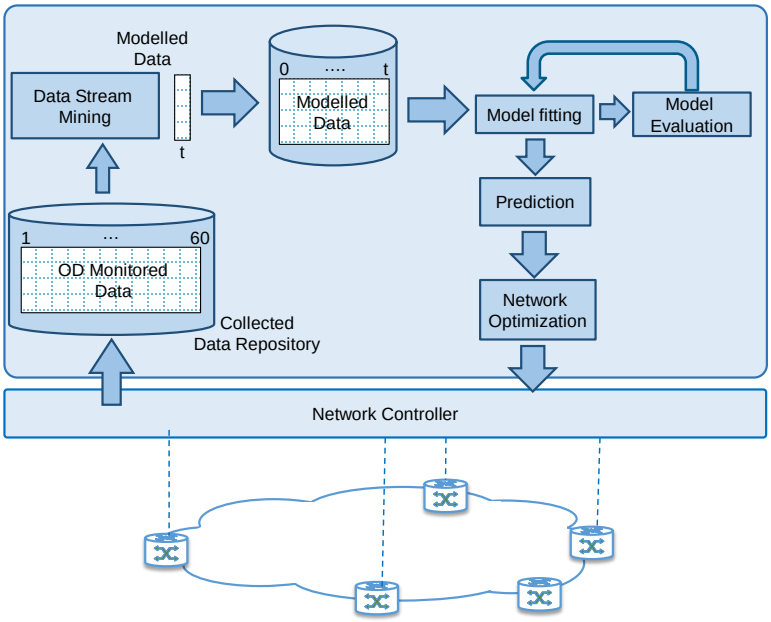

Figure 2-2: Big Data Backed telecom clouds scheme

Traffic is generated by users through different services such as mobile applications and residential or business connections. An origin node aggregates traffic from a group of users and sends it to the destination node in the network. We assume that every node monitors OD traffic by collecting a set of samples and sends it to a centralized module that stores it in a *Collected Data Repository*. As we are considering OD traffic, $|N|(|N|-1)$, where N is the number of nodes, samples are stored at every monitoring interval, as we are considering OD pair's traffic.

Periodically, e.g. every hour, the collected data for a given OD pair in the *Collected Data Repository* is summarized applying data stream mining, producing a *Modelled Data Repository*. This modelled data contains the minimum, maximum, average, the last value of each of the predefined period, and a time stamp.

After a predefined number of modelled data periods *Model fitting* is performed. In this project we propose different types of models for this module. *Model evaluation* is carried on the models, and if a model is considered to be valid, it is used to predict future OD traffic. The predictions of future traffic for OD pairs are used by a *decision maker* module to decide if the network needs to be reconfigured or not. If it is the case, the traffic predictions are used to find the optimal reconfiguration. The Network Controller is in charge of modifying the network appropriately.

Thanks to this architecture, the network can be optimized by adapting its topology and capacity to future changes on the traffic that would lead to poor performances of the network otherwise. Further details can be found in [Mo16].

2.4 OMnet++ network simulation environment

OMNeT++ [OM] is an extensible, modular, component-based C++ object-oriented discrete event network simulation framework. It provides a component architecture for models, that are programmed in C++ and assembled into larger components and models using high-level language (NED) defining a topology.

Throughout this project, the different algorithms, generation methods and modelling are implemented and embedded in an OMNeT++-based simulator. This simulator emulates the real performance of an optical network. It is developed in OMNeT++ 4.5 using C++11 with BGL 1.53.0 and Xerces-C++ 3.1.1 libraries. The simulator is organized in a number of modules representing either logical or physical elements in optical networks.

Each module has been implemented to simulate several functionalities. Its combination allows emulating the architecture and the protocols of a network. Different optical network topologies can be created specifying a configuration of nodes in a .ned file.

Each node is able to generate traffic according to a methodology that is described in Chapter 4. The explanation of other modules follows in that chapter, which are in charge of analyzing the provided traffic and producing from it models that are later used to predict the network traffic, and make decisions according to these predictions.

2.5 Conclusions

In this chapter we have introduced the optical transport networks and how it can benefit from data analysis with the Big Data backed telecom cloud architecture. The several models and implementations that are proposed in the following chapters serve to give further capabilities to this architecture. To assess these, the OMNeT++ simulator is the framework on which models and processes are implemented and tested.

The following chapter explains in depth the modelling methodology that has been used to obtain the models of the project.

Chapter 3.

Modelling methodology

Throughout this project we consider traffic defined as a general time series, that is, a series of values with its relative time of when the value was taken. This section gives a background on the concepts and methods related to time series that are used along the project. It also gives a background on Neural Network Models and their use to model time series; all of these topics are based on [Ad13]. Modelling seasonal component by polynomial fitting is explained next. Different methods to evaluate the goodness-of-fit of different models to a given data set are discussed. Finally, different methods implemented during the project, to calculate related components to model data series are explained in detail.

3.1 Time Series Background

A *time series* is a sequence of data points ordered by the time each observation was taken, assuming that the time is spaced at uniform intervals. To generalize this for

$$(v(i), t(i)), \text{ with } i = 1,2,...,n \tag{3.1}$$

non-uniform time intervals, we use the following notation to describe a time series:

Time series have three important components:

- *Trend* T_t: long term tendency of a time series to increase, decrease or remain flat.
- *Seasonality* S_t: oscillations within a fixed period of time that can be observed throughout the data. This is due, for example, to correlation with time of day, week, month or year.
- *Random variations* r_t: caused by unpredictable or unmeasured variables.

In a time series data set, the variable v fluctuates randomly dependent with time and with two main deterministic components: the trend and the seasonal component. The trend takes account of some long term evolution of the data series, and the seasonal component takes account of some period that repeats itself throughout time due to the dependency of the traffic. The random variations produce nondeterminism.

The study of time series aims to explain the process that generated the data and its properties to find models to make predictions of future behavior. With the above three components, we can model the data series by the *additive model*:

$$v_t = T_t + S_t + r_t,\ t = 1,2,...,n \tag{3.2}$$

Or, via logarithmic transformation, by the *multiplicative model*:

$$v_t = T_t S_t r_t,\ t = 1,2,...,n \tag{3.3}$$

The joint distribution function that generated the time series can be described by the first and second moments:

- *Mean* $\mu_t = \mathrm{E}(v_t)$
- *Variance* $\sigma_t^{\,2} = Var(v_t) = \mathrm{E}((v_t - \mu_t)^2)$
- *Autocovariance* $\gamma(t_1,t_2) = Cov(v_{t1}, v_{t2}) = \mathrm{E}((v_{t1} - \mu_{t1})(v_{t2} - \mu_{t2})), \forall t_1 t_2 \in T$
- *Autocorrelation* $\rho(t_1,t_2) = \dfrac{Cov(v_{t1}, v_{t2})}{\sqrt{Var(v_{t1})Var(v_{t2})}}$

The *Autocorrelation Function* (ACF) of the series gives the correlation between v_t and v_{t-h} for h=1,2,3… The sample lag-h autocorrelation is given by:

$$\rho_h = \rho(v_t, v_{t+h}) = \frac{\gamma_h}{\gamma_0} = \frac{\gamma_h}{\sigma^2}, \qquad \text{where } \gamma_h = Cov(v_t, v_{t+h}) \tag{3.4}$$

The *Partial Autocorrelation Function* (PACF) gives the partial correlation of a time series with its own lagged values between v_t and v_{t-h} for h=1,2,3…, with the linear dependence of v_t and v_{t+k-1} removed, that is, the autocorrelation that is not accounted for by lags 1 to k-1, inclusive. The partial autocorrelation of lag k is given by:

$$\alpha(1) = Cor(v_{t+1}, v_t)$$
$$\alpha(k) = Cor(v_{t+k} - P_{t,k}(z_{t+k}), v_t - P_{t,k}(z_t)) \tag{3.5}$$

where $P_{t,k}$ denotes the projection of x onto the space spanned by $x_{t+1},...,x_{t+k+1}$.

A series is called to be *stationary* if the process that generated it had a joint probability distribution that does not change when shifted in time. That is, properties such as mean, variance or autocovariance remain constant through

time. It would be expected for a given time series to be stationary after the trend and seasonality components have been eliminated. We proceed to explain some methods to preprocess data and obtain new stationary series from any given series.

Usually a time series is heteroscedastic, that is, variance of the noise depends on the scale of the series. For example, if the series has an increasing trend, the variance also increases with time. Therefore, usually the logarithm is first applied to stabilize the variance, obtaining a new homoscedastic series, and afterwards further preprocessing is performed.

The second preprocessing that needs to be performed is the extraction of the tendency. The trend component for can be easily obtained by applying curve fitting to data samples to identify some well-known traffic evolutionary pattern, e.g., constant, linear, quadratic, or exponential. The one producing the smallest error in terms of some measure, such as maximum squared error, is selected as trend function.

After removing the trend effect from input data, the next step is to compute the period the resulting series has, to be later able to produce a periodic model with that seasonal component. Useful methods to detect seasonality study the ACF or the PACF, since these functions will show high correlation between the observation at time t and the observation at time $t-p$, where p is the period of the series.

To apply certain statistical models, further preprocess has to be performed on the data. Generally, we can talk about *filtering* a time series, which consists on calculating a linear combination of the observations in order to obtain another time series:

$$w_t = \sum_{j=-q}^{q} a_j v_{t+j} \text{ , where } q \in Z^+ \text{ and } a_j \text{ constants} \tag{3.6}$$

First order differentiation is also a linear filter defined as:

$$\nabla v_t = v_t - v_{t-1} \tag{3.7}$$

If the process has a deterministic linear trend plus a noise with mean zero, applying a first order differentiation we obtain a process with no trend and constant mean.

To remove a serial dependency to a given lag k, differentiation of bigger order can be applied, by converting the ith element of the series into the difference of itself minus the element at kth position. Therefore a differentiation of lag p, where p is the period of the series, is a way to obtain a new series with no seasonality.

Once tendency and seasonality components are removed, the remaining component is the random variation. If no further dependencies exist on the data the random noise is expected to be Gaussian noise (noise with probability density function equal to that of the normal distribution). When this is not the case, further modelling such as ARIMA can be applied to fit the residuals.

3.1.1 ARIMA

One of the most popular stochastic models used in time series is the *Autoregressive Integrated Moving Average* (ARIMA) model. It is a combination of two simpler models and preprocessing of data.

First of all we are going to consider the simplest model, which is the *autoregressive model* (AR), which specifies that the output variable depends linearly on its previous values and on a stochastic term. The autoregressive model of order p, AR(p), is defined as:

$$v_t = \varepsilon_t + \sum_{i=1}^{p} \varphi_i v_{t-i} + c \tag{3.8}$$

Where φ_i are the parameters of the model, c is a constant, and ε_t is white noise (random component with the property that its value does not have correlation at any given two times).

Secondly, the *Moving Average* (MA) is a filter which creates a series of averages of different subsets of the full data set. It is used to eliminate the seasonality component and highlight the longer term trend. It follows the formula:

$$M_t = \frac{1}{2q+1} \sum_{j=-q}^{q} v_{t+j} \text{ , where } q \in Z^+ \tag{3.9}$$

To model non-stationary series, the series is reduced to stationarity by suitable differentiation and the resulting differenced series is then modeled by an ARMA process, which is a combination of a MA and an AR. To apply this method, the time series has to have certain properties: variable independent and identically distributed following a uniform distribution, stationarity (the joint distribution of any possible set of random variables has to be independent from the time) and non-seasonality (no seasonal component should be observed in the data).

Both stationarity and non-seasonality can be obtained preprocessing the data by differentiating it (subtract at each observation the observed value at the same time of the previous period).

After the preprocessing, the best ARIMA(p,0,q) which is equivalent to an ARMA(p,q) model is fitted into the data. In R one can find the best parameters (p,d,q) with the function auto.arima, which compares different combinations of parameters' accuracy and parsimony. ARMA(p,q) is a combination of an AR model and a MA model, and can be therefore formulated as:

$$v_t = \varepsilon_t + \sum_{i=1}^{p} \varphi_i v_{t-i} + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i} \tag{3.10}$$

3.1.2 Artificial Neural Networks

Artificial Neural Networks (ANNs) are a family of models in Machine Learning inspired by biological neural networks, which are used to approximate functions that can depend on a large number of input variables.

An ANN is formed by two or more layers joined by weights. The first layer is the input layer, which has a neuron for each input variable the model has. The last layer is the output layer, which has a neuron for each output variable the model has. These two layers can be connected through a number of hidden layers, which can be themselves of different number of layers and structures, formed by the hidden neurons. The input variables are connected, therefore, to the output variables through a number of weights that are trained through known sets of values of the variables, making the neural nets adaptable and with a similarity to the learning process.

Figure 3-1, shows a scheme of a general ANN. It takes n input variables, and m output variables. The hidden neurons are distributed in r hidden layers, each of them with t_i neurons, $i = 1,\ldots,r$. The n input lags are connected to the first hidden layer through the weights $w_1(i,j)$, $i = 1,\ldots,n$, $j=1,\ldots,t_1$. Also an intercept neuron is connected to the first hidden layer through the weights $w_1(0,j)$, $j=1,\ldots, t_1$. The remaining hidden layers are connected to the layer behind each of them through the weights $w_h(i,j)$, $i = 1,\ldots,t_{h-1}$, $j=1,\ldots,t_h$, and also to its intercept through the weights $w_h(0,j)$, with $j = 1,\ldots,t_h$. The last hidden layer r is connected to the output variables through the weights $w_f(i,j)$, $i = 1,\ldots,t_r$, $j=1,\ldots,m$, and the output variables also receive an intercept through the weights $w_f(0,j)$, $j=1,\ldots,m$.

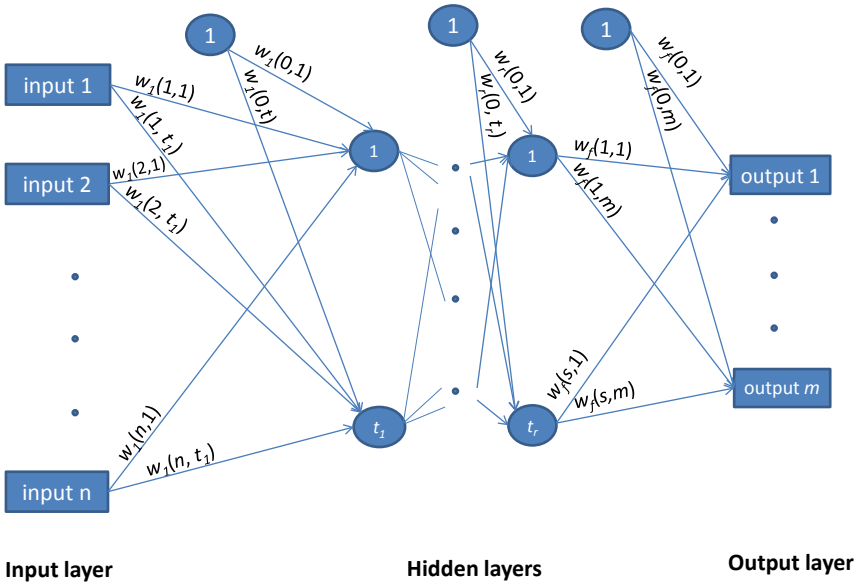

Figure 3-1: ANN scheme

The output of each neuron is a function of the outputs of all the neurons in the previous layer and the intercept. The *j*th neuron of the *s*th hidden layer would give as output:

$$n_{s,j} = f\left(\sum_{i=1}^{t_{s-1}} f(w_s(i,j)n_{s-1,i})\right) + w_s(0,j) \tag{3.11}$$

where f is the activation function $f(x) = \frac{1}{1+e^{-x}}$

We are here using the logistic function as the activation function, but other functions can be used, such as the sigmoid function or the hyperbolic function.

The function producing one of the output variables of the model displayed in Figure 3-1 is the following:

$$output_j = f\left(\sum_{i=1}^{t_r} f(w_f(i,j)n_{r,i})\right) + w_f(0,j) \tag{3.12}$$

First the structure of the ANN is determined, that is, the number of input and output variables and the number and the way they are distributed into layers of the hidden neurons. Then this given structure is trained with some data, which means finding some optimal values of these weights, w_i, that will, from the input variables, produce the minimal error by some predetermined cost function, output variables. To do the training, several algorithms exist, for example gradient descent backpropagation, Levenberg–Marquardt backpropagation or the genetic algorithm.

Artificial Neural Networks deal well with the linear limitations of models such as ARIMA, and also need no assumption on the distribution that needs to follow the observations, only rescaling the data is sometimes necessary depending on the activation function. It has been shown that ANN with just one hidden layer, result known as the universal approximation theorem [KuH], can approximate any given continuous function to any desired accuracy, so in the implementations carried through this project only one hidden layer is considered.

When dealing with ANN one has to be very careful about many decisions:

- choice of input variables
- choice of architecture (number of hidden units) and activation function
- criterion for selecting the final model (both accuracy and parsimony should be considered, we want good fitting of data without overtraining the model to avoid over-fitting problems)
- knowing if data needs to be rescaled (some activation functions require normalized values)

- choice of starting values of weights (there can be several local minimums because ANN are non-deterministic).

In our case, we are interested in modeling a time series through an ANN. To that end, we consider only one output variable, v_t, and the input variables are previous observations of the output variable associated time: v_{t-1}, v_{t-2},…, v_{t-n}, [Fa98].

Seasonal time series do not require any preprocessing of the data either. The network learns the seasonal component without need of removing it. If the seasonal period is known, one can use this to determine the number of previous observations needed as input variables (the observed value of the variable from time t-p and to t-1 can be given as input variables to predict the variable at time t).

A structure is determined for the neural network, and the training of the model is carried, obtaining some final weights for the ANN. Training is be done with the training set, in this case, some already observed part of the time series.

3.2 Polynomial fitting of periodic patterns

Assuming we have a homeostatic time series and that it has no trend, we are next interested in extracting the seasonal component, so that the only remaining nondeterministic component is the remaining noise. Differentiation is the preprocess usually used in ARIMA, but one could also extract the seasonal component by computing an average profile of the period length with the first observed values of the series, and subtracting each average to all time series. To that end, if the time series has a period *per*, we propose a model that would normalize the times *t[i]* by *per* via a modulus operation, and to the obtained discrete values *(v[i],t'[i])* with *t'[i]* between 0 and the *per*, fit a polynomial to obtain a periodic and continuous model.

If the observations are taken every hour, when we normalize by the period we have as many values *v[i]* as the number of periods there are in the part of the time series considered. Observe that if we fitted a polynomial of degree higher than the period length in the least squares sense, this would give us a profile in which for the observed times, the average of the values that have the same time after normalization by the period would be given: the problem is that obviously overfitting will occur, and the prediction for times where we had no observation will be bad. Also it would be computationally expensive to fit a polynomial of such a large degree.

We need to find a way to decide the optimal degree of the polynomial, the one that fits well the discrete points in a way similar to the mean for the previously observed times, but does not overfit the data, providing also a good prediction for not previously observed times. We propose to decide this by comparison on the BIC for different degree polynomial models fitting the data, dealing thanks to this, with

both accuracy and parsimony, i.e. producing a good fit to the data while avoiding overfitting. The degree associated with the minimal BIC is obtained and a polynomial of this degree is fitted into the normalized data, producing a continuous profile for the seasonal component.

Prediction of a future week could be made by simply observing the average value of the past periods for each given time. If there is no trend, only the random variances would produce errors in the prediction.

3.3 Model Selection and Validation

We need ways to compare the goodness of fit of different models to some given data set to choose the one with best predictive accuracy. To do so, there is the need of computing in some way the prediction or fitting errors of the model. Also complexity of the model has to be somehow computed in order to be able to compare fairly between different models. It is important to have accuracy, but also avoid highly complex models that will likely produce overfitting on the data and therefore big predictive errors when encountering not previously seen sets of data.

3.3.1 AIC and BIC

When modeling data, one has to be especially careful with two things: accuracy and parsimony (introduced in section 2.2). A good model needs to have minimal errors to the given data (accuracy), but at the same time minimal possible number of parameters to avoid overfitting (parsimony). Therefor criterions for model selection need to be measures that compare goodness-of-fit of the models but with a term that disadvantages a model as its complexity increases. *Akaike Information criterium (*AIC), and *Bayesian Information Criterium* (BIC), are example of this, and both provide means for model selection [Ka95].

The AIC is a measure of the relative quality of statistical models for a given set of data. It is not a test in the sense of testing a null hypothesis so it does not give a value in an absolute sense, but it can be used to compare different models used to fit the same dataset and choose as the best model the one with smallest AIC. The AIC value of a model is the following:

$$AIC = 2k - 2\ln(L) \tag{3.13}$$

Where L is the maximum value of the likelihood function and k is the number of parameters of the model. The minimal AIC of different models will be the one that has small errors, assessed by the likelihood function, but not too many parameters, as there is a penalty on the number of parameters.

If we assume that the residuals are i.i.d. N(0,σ), the maximum likelihood estimate can be obtained through the residual sum of squares RSS:

$$AIC = 2k + n\ln(RSS) \tag{3.14}$$

where $RSS = \sum_{I=1}^{n} (y_i - f(x_i))^2$.

The BIC is also a criterion for model selection among a finite set of models, again based on the likelihood function and a penalty term that increases as the number of parameters increases.

$$BIC = -2\ln \bar{L} + k\ln(n) \tag{3.15}$$

Where n is the number of data points, k is the number of parameters of the model and $\bar{L}$ is the maximum value of the likelihood function. Which again, under the hypothesis of i.i.d. N(0, σ) residuals can be written as:

$$BIC = k\ln(n) + n\ln(RSS/n) \tag{3.16}$$

3.3.2 Cross-validation

Cross-validation is a model validation technique for assessing how the results of a statistical analysis will generalize to an independent data set.

The idea of cross-validation is to partition the data set into two disjoint groups: the training set and the cross-validation set. The statistical analysis or the model will be computed only using the training set, and the assessment of the goodness of fit of the model will be performed on the cross-validation set. By doing this we can estimate how well the model will do when it encounters sets of input data that had not been seen before. Since the two sets are disjoint and appropriately separated, overfitting will also be easy to detect as the errors on the testing set will be big.

To the cross-validation set, measures of errors such as the RSS (Residual sum of squares) can be used in order to assess the prediction performance of the model.

3.3.3 Hypothesis test

We already explained criterions to select the best model between a finite number of possible models. Traffic is, though, usually evolutionary, and this may mean that after a time space, the selected model will no longer be able to produce accurate predictions. To that end, a χ^2 hypothesis test can be used to decide if a model is still valid or not.

The statistic for the hypothesis test is:

$$\lambda = \sum_{i=1}^{n} \frac{(o_i - e_i)}{e_i} \tag{3.17}$$

Where o_i is the ith observed value in the series, e_i is the prediction of this value by the given model and n is the number of observations we are considering.

The p-value will indicate that the model is still valid, when the p-value with $n-1$ degrees of freedom and statistic λ is bigger than a threshold (usually 0.05) and that the model is not valid when the contrary is fulfilled.

3.4 Proposed algorithms

Up to now different models have been proposed, and some of them required some computations of components or preprocessing. We next propose our implementation to compute these components and preprocessing.

3.4.1 Standard deviation

It is of great use to have the standard deviation of the original time series, to obtain from it a confidence interval for the predictions. To obtain this, the standard deviations of time series are calculated, with the proposed algorithm in Table 3-1. As discussed before, heteroestaticy is a usual characteristic of a time series, and this is taken into consideration.

Table 3-1: computeStDev

1:	Given a time series *(v[i],t[i]), i*=1..*n*
2:	**for** k= 0..numPartition
3:	yPart ← traffic intensity partition
4:	**end**
5:	**for** k= 1..numPartition
6:	stdev[k-1] ← stdev of {v[i] s.t. v[i] ∈ (vPart[k-1],vPart[k])}
7:	vres = (0.0,stdev[k-1]), k s.t. v[0] ∈ (vPart[k-1],vPart[k])
8:	vres = concatenate(vres, ((t[i]+t[i-1])/2 , stdev[k-1])), k s.t.
9:	v[i] ∈ (vPar[k-1],vPart[k]) and v[i-1] ∉ (vPart[k-1],vPart[k])
10:	**end**
11:	**return** vres

To increment accuracy of the confidence intervals, the range of values that takes v is first partitioned into *numPartition* groups (line 2), and for each of the groups, the standard deviation is calculated (line 4). The result of the algorithm is a pair of two vectors, the first containing the different times where v changes of range partition, and a second vector containing the associated time interval's standard deviation. The standard deviation at a given time t can be calculated as a function of *t, st(t),* by finding the range partition in which t belongs to and getting its associated standard deviation.

3.4.2 Trend and homoestecity

As stated before, standard deviations may depend on the scale of the traffic, and this means that series can not only have an additive trend, but can also be hetereostecit. What we propose next is a revertible method to obtain a new series from the original with homeostacity and no trend. Extracting these components from the data set obtaining a stationary in mean series to which we are able later to produce predictive models.

We first make sure that there is no change on intensity due to a tendency multiplicity factor, which makes the time series to increase or decrease not only in value but also in width (hetereostecity), as an alternative to the preprocessing by taking logarithms.

We propose the following algorithm to compute this variation, and produce a change on the traffic so that the intensity width remains the same.

Table 3-2: getHomoestecity

```
1:   Given a time series (v[i],t[i]), i=1..n
2:   Calculate period p of (v,t)
3:   for nperiods=1 ..n/p
4:       maximum = max{v[i]:i in (nperiods-1) ·p,...,nperiods ·p}
5:       minimum = min{v[i]:i in (nperiods-1) ·p,...,nperiods ·p}
6:       width[nperiods] = maximum - minimum
7:       relative_width = width[nperiods]/width[1]
8:        times[nperiods] = t[nperiod]
9:   end
10:  (a,b) = linear_regression(times, relative_width)
11:  for i in 1..n
12:      vnorm[i] = v[i]/(at[i]+b)
13:  end
14:  return (vnorm[i], t[i]), i=1..n
```

For each seasonal oscillation in the series, we compute its width by taking its maximum and minimum values (lines 4,5 and 6), and normalize it by the first computed width. To the series of relative width a linear regression is performed (line 10), and from the obtained function, a new series *vnorm* that satisfies homeostacity is obtained via the formula in line 12. Observe that if the series was already homeostatic we would expect $(a,b)=(0,1)$, while, if the series had an incremental multiplivative tendency, (a,b) would have a value of a bigger than 0.

This newly obtained time series *(vnorm[i],t[i])* can still contain a trend in the additive sense. We are also going to compute it and extract it in order to have a fully stationary time series.

Table 3-3: getTrend

1:	Given a time series *(v[i],t[i]), i*=1..*n*
2:	Calculate period *p* of *(v,t)*
3:	**for** *nperiods=1...n/p*
4:	mean[*nperiods*] = mean{*v*[i]: i in (*nperiods*-1),..,*nperiods*}
5:	times[*nperiods*] = t[*nperiod*]
6:	**end**
7:	*(c,d)* = linear_regression(times, mean)
8:	**for** i in 1..n
9:	*vnorm2[i] = v[i] - ct[i] – d*
10:	**end**
11:	**return** *(vnorm2[i], t[i]), i*=1..*n*, and *(c,d)*

In this case we proceed again to consider each seasonal oscillation, and compute its mean (line 4). Next a linear regression is applied upon the means, and through this a new series *vnorm2* with no additive tendency is obtained via the formula at 9.

We are assuming the trend is linear, but this could be generalized to any other trend by fitting another function to the data (at line 7).

3.4.3 Series period computation

Observe that the previous algorithm needed the period of the data set. It can be obtained by studying the ACF and PACF of the series, but we are interested in automatizing this computation. To do so, we have implemented an algorithm, explained in Table 3-4, which computes the period of a series.

Table 3-4: Compute period

1:	Given a time series (v[i],t[i]) i=1..n, rsd, params
2:	R←∅
3:	**for** p_t in params.range **do**
4:	S ← splitInChunks(v,t)
5:	computeAverageAndSlope(S)
6:	a←computeVarianceAverage(S)
7:	b←computeVarianceSlopes(S)
8:	$R(p_t) = ab/\sqrt{S.length}$
9:	**end**
10:	**if** stddev(R)<params.β
11:	**then return** inf
12:	**end**
13:	P ← {i | R[i]<= params.α * min(R)}
14:	**return** min(P)

The idea of the algorithm is the following. Given a time series *(v,t),* all candidate periods are sequentially explored, in a range between a minimum and maximum in steps of a fixed duration, e.g., 1 hour (line 3 in Table 3-4).

For each candidate period size p_t, the input data is split into chunks of length i containing samples consecutive in time (line 3). Different starting times are considered, and to all of these chunks, a linear regression is applied to the points it contains. We are interested in the slope and the mean of this linear regression for the following reason: if the candidate period size is actually the real period length of the series, then all fitting lines will have a similar slope and a similar mean, and if it is not the case, then the fitting lines will have different slopes and means. Smaller deviations between the subgroups of a given possible period imply that the probability of the series to have that period increases.

To be able to give an estimator of how similar the means and the slopes are, the standard deviations of these means and slopes are calculated for all chunks obtained from the same candidate period. It is natural to think that the real period of the data series will be the one that has smallest standard deviations in both means and slopes. The estimator R that will be used to compare the different tested periods, p_t, is given by the product of the standard deviations of means and slopes, normalized by the number of chunks considered.

Now it is important to notice two problems. Data might have two periods, for example the daily period and the weekly period, but we are interested in the longer one. For this reason it does not suffice to stop the algorithm once a local minimum estimator is detected, since this would lead to the daily period. Also, a weekly period implies a two, three and so on week periods, which means that our estimator will be similarly small for one, two, three and so on week periods. For this reason it does not suffice either, to take the overall minima of all estimators, since this might lead us to, for example, a three weeks period, when the real period is one week.

Examples of these problems when fitting different period candidates can be observed in Figure 3-2. (a) shows how the fitting of the period candidate 24 hours would look like. Beginning at time t=1, the linear regression for each 24 hour chunk is performed. From these linear regressions, the standard deviation of the mean and the slopes is computed, and the same is done for other starting times. Notice that the slopes for the linear regressions are quite similar because of the daily period, but the mean will have bigger standard deviations because of the difference between week and weekend days.

Figure 3-2 (b) shows how the fitting of the period candidate 168 hours would look like. In this case, since the period being tested is the actual period of the data series, notice that the linear regressions for the different chunks will give similar means and slopes, and therefore the associated estimator will be smaller than the associated estimator for the other period candidates.

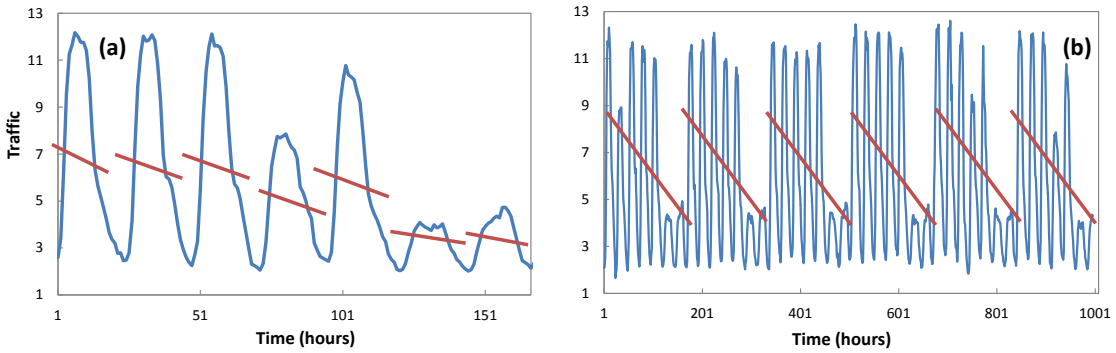

Figure 3-2: (a) 24 hours period fitting (b) 168 hours period fitting

To put a solution to these two observed problems, the final period solution is chosen from a percentage of the smallest estimators, and from them choosing the smallest possible period.

3.5 Conclusions

Throughout this section, general models for modeling time series have been introduced: ARIMA, Neural Networks and Polynomial fitting. Next, selection and validation methods for given models of the time series have been discussed. From all these techniques, the need of some specific algorithms raised. And to that end, our implementation of methods to preprocess data (make it homoestatic and extract the trend), compute some characteristics (standard deviations and period of the data set) has been explained.

It is important to clearly understand what kind of preprocessing every model needs in order to be applied, and which of the implemented methods can be used to obtain this preprocessed data series. Figure 3-3 shows a scheme of how the needed preprocessing for each of the considered models is used and the resulting model obtained.

All models start with a general non-stationary seasonal time series. ARIMA applies differentiation twice to obtain a stationary non seasonal series, to which an ARMA model is fitted. ANN do not need further preprocessing other than the normalization of the traffic in order to apply the activation functions, and to it the training of the ANN structure is performed. As for the proposed Polynomial model, a series satisfying homoestaticy and with no tendency can be obtained by the implementations discussed in 3.4.2 and also the period can be computed as discussed in 3.4.3; the time is normalized by the computed period, and to the newly obtained discrete values a polynomial is fitted to model the seasonal component of the series.

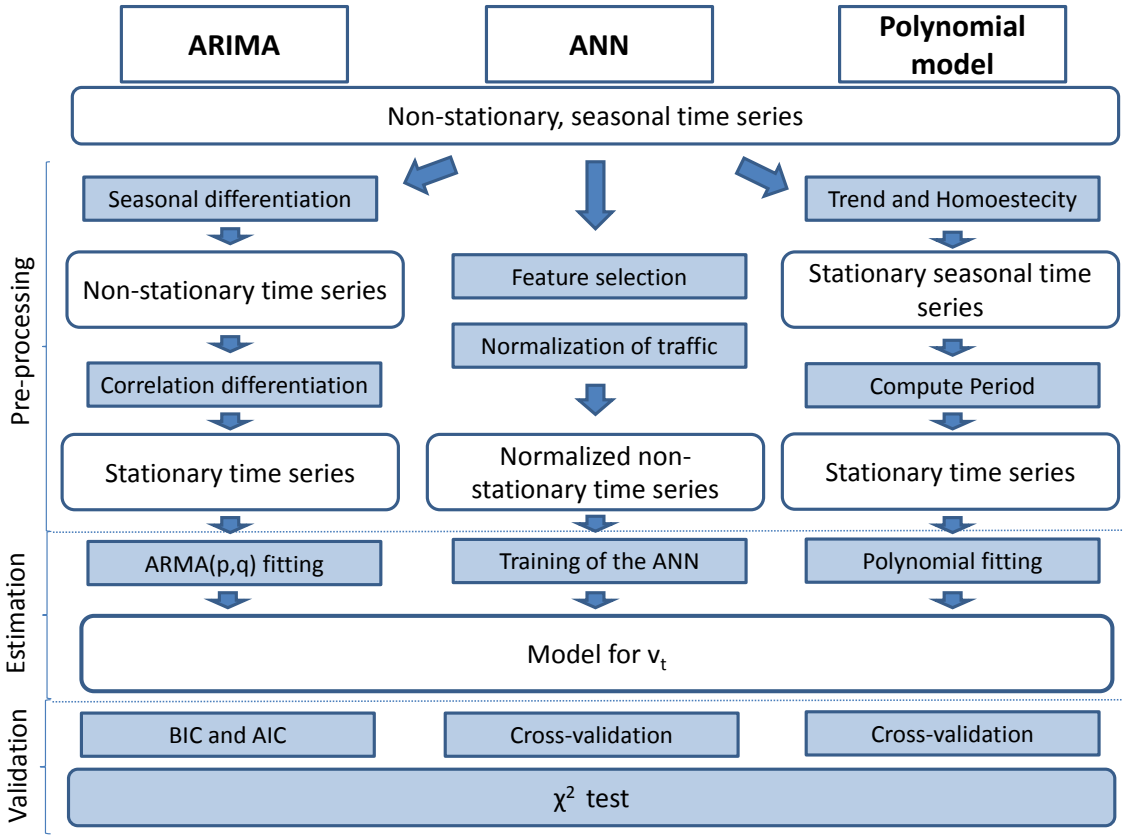

Figure 3-3: Models application scheme

When choosing among a finite number of models we proceeded to the use of model selection methods. Usually ARIMA is related to the use of BIC, AIC and hypothesis testing such as the Chi-squared test, while ANN uses cross validation. As for the polynomial model, we already used the BIC for the choice of the degree, and cross-validation can be used to study the remaining residuals.

Although the described procedure is the usual one for ARIMA and ANN, a combination of the procedures might be a good approach. This combination allows more flexibility during the generation. For instance, having a model for the seasonal component with the polynomial model, and to it adding the remaining noise, that can be simply modeled as an i.i.d $\varepsilon_t \sim N(0,\sigma^2)$, or a more accurate noise that can be obtained by fitting an ARIMA or an ANN to the noise, will prove to be very useful for the generation of traffic, which is studied in Chapter 4.

Also a combination of the methods will be useful when modeling a given data series to predict its future behavior. We could fit the stationary time series obtained as a noise of the polynomial model with an ARMA or an ANN and compare the results of these models with the ARIMA applied by usual differentiation or the ANN applied to the original time series. This is what is done in Chapter 5.

Finally, the small pre-processing needed to fit an ANN but at the same time the accuracy in which these models are able to predict the future of a time series, will prove to make ANN the most useful predictive model, as is seen in Chapter 6.

Chapter 4.

Implementation

This section explains how we have integrated in the OMNET++-based simulator realistic traffic functions to generate network traffic, and how traffic is used to produce models that are able to predict its future behavior. The traffic generation is necessary to be able to study the network, algorithms and models performances in the simulator, as real traffic is not available.

4.1 Traffic Generation

Our main goal is to be able to generate general traffic profiles. Denoting this traffic profile as *f(t;T),* a function dependent on the time *t* and with period *T*, we express it as a combination of simpler functions to enable better understanding of the components the traffic has and more flexibility in the generation and change of traffic properties.

We first need to define the simplest functions that participate in the generation process of the simulator, the *Unitary functions*, which will serve as a base for more complex functions, and are defined in the simulator as OMNeT++ modules. These functions are contained in the area defined by corner-points (0,0), (1,0), (0,1) and (1,1). Of our interest are the following: the piecewise linear, and the polynomial unitary functions.

There are also *Distribution functions*, which are generated with a random seed, and are constrained with y values from 0 to 1 and are also defined as OMNeT++ modules. From this we will typically use normal distribution function that will allow us to add normally distributed randomness to the generated traffic.

From these functions, more general traffic satisfying the desired properties depending on the traffic that wants to be generated, can be produced through what we will call *Custom functions*, and are defined also as OMNeT++ modules. The idea behind any Custom function is to take a Unitary function or a Distribution function

and impose to it the desired period T (it will usually be 24 hours, so that data has a daily periodicity), and some intensity scale. From this, we have: the scaled polynomial, scaled piecewise linear and the noise (which is a scaled normal distribution) custom functions.

Another important Custom function is the correlation function, which makes traffic dependent to its past values (we see in Chapter 5 how this is the case for real traffic data series). To that end, correlated traffic is generated by the following formula:

$$traffic(t) = f(t) + \sum_{i=1}^{k} c_i f(t-1) \tag{4.1}$$

Where *f(t)* is the value of the profile traffic for that time and to it a linear combination of the past observed values is added.

Usually generation of correlation traffic is done via the following formula, where a noise is added to the previous one:

$$traffic(t) = f(t) + \sum_{i=1}^{k} c_i f(t-1) + \text{noise(t)} \tag{4.2}$$

The last important Custom function is the alpha function α, which follows the formula:

$$\alpha(t) = 1 - \frac{t}{t_{final}} \tag{4.3}$$

where t_{final} is the final simulation time. This allows us to smoothly change from one profile, p_1, to another profile, p_2, as time advances in the simulation, by a formula of the type:

$$traffic(t) = \alpha(t) p_1 + (1 - \alpha(t)) p_2 \tag{4.4}$$

The output of Custom functions is connected to the traffic combiner, an OMNeT++ module which allows combining different Custom functions to produce traffic. The combination is done as an addition, subtraction and multiplication of different Custom functions, and is defined as an input string. The previous traffic would be defined by the following string:

$$"\_Alpha\_*(\_F\_+\_Noise\_)+(1-\_Alpha\_)*(\_G\_+\_Noise\_)" \tag{4.5}$$

where _F_ and _G_ would be two custom functions defining the two profiles of the series, _Alpha_ would be the α custom function, and _Noise_ the *noise(t)* uniformly distributed residuals.

The overall structure of the different modules that participate in the generation of traffic can be observed in the scheme shown in Figure 4-1.

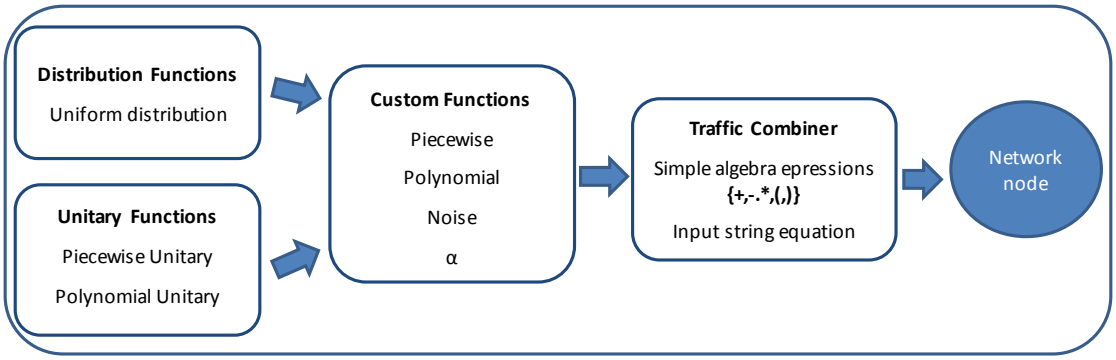

Figure 4-1: Design of a flexible framework to model traffic generation

Traffic is generated in a module of the simulator, called Analytics, at small intervals of time from combinations of Custom functions. The combination of all these modules, gives us a lot of flexibility to generate different types of traffic, with different properties, degrees of randomness and correlation, and with different evolution throughout time.

The basic profiles that are used later in this project are a business-like traffic, B, with differentiated traffic intensities during working and night hours, with an increase at 9 in the morning and the maximum at 12. The second traffic profile that is used is the one related to a content delivery network (CDN), which is traffic generated by residential users, with a maximum value around 17h.

The first traffic that we consider is an static traffic (with no trend and no change in amplitude), produced by a sum of a business daily profile, *B(t),* defined by a piecewise linear custom function with period 24 hours and scale 1500, and a noise, *noise(t),* defined by the noise custom function, following a N(0.5,3) and scaled at 10:

$$traffic(t) = B(t) + noise(t) \tag{4.6}$$

Where *B(t)* would be the seasonal component S_t, and *noise(t)* would be the random component r_t, following the notation used at 3.1. Figure 4-2 shows how one week of this traffic looks like, plotting the traffic versus the time in days. The CDN traffic can be generated in an analogous way by another piecewise linear function.

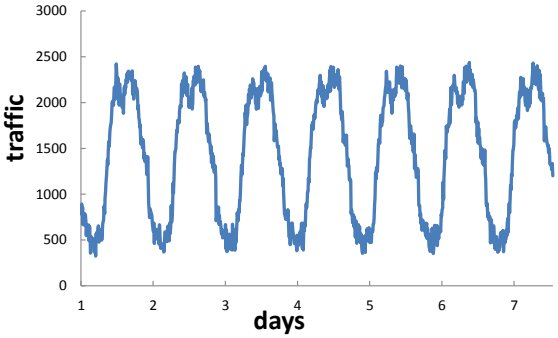

Figure 4-2: Simulated business traffic

In case we are interested in traffic that evolves in the long term, that is, it either has a trend, or it changes from one profile to another, we can generate this as we said before via the α function or via the addition of a trend Custom function.

First of all we consider a traffic with increasing intensity, that is, starting at the initial daily profile, *B(t),* and linearly evolving (with the Custom function α) to end, after 40 days at a 5*B(t)*. This is done generating traffic with the following formula, and can be observed in Figure 4-3.

$$traffic(t) = \alpha(B(t) + noise(t)) + (1-\alpha)(5B(t) + noise(t)) \tag{4.7}$$

Where B is defined as before, *noise* is a normal distribution error function of parameters N(0.5,3) and multiplied by a 10 intensity factor, and α is the Custom function with parameter t_{final}=40 days.

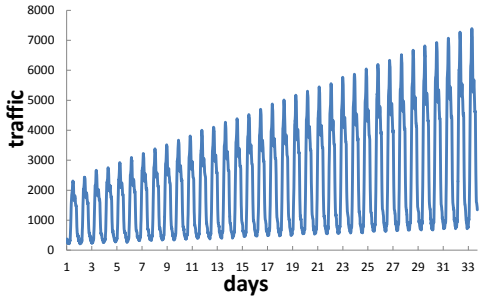

Figure 4-3: Simulated traffic with intensity change

We are now going to consider traffic that linearly evolves, in 15 days, from the business profile to the CDN profile. This traffic is obtained by the following formula, and plotted in Figure 4-4.

$$traffic(t) = \alpha(B(t) + noise(t)) + (1-\alpha)(CDN(t) + noise(t)) \tag{4.8}$$

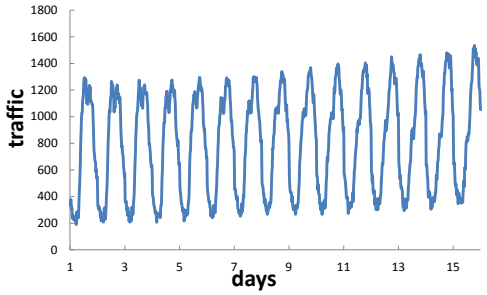

Figure 4-4: Simulated traffic with profile change

Finally we can combine both evolutions of traffic, a change on intensity and profile. To do so, we produce a 40 day traffic, that starts with profile traffic *B(t)* with intensity 1, and ends with a profile traffic *G(t)* with intensity 5. This traffic is generated with the following formula, and can be observed in Figure 4-5:

$$traffic(t) = \alpha(B(t) + noise(t)) + (1-\alpha)(5CDN(t) + noise(t)) \quad (4.9)$$

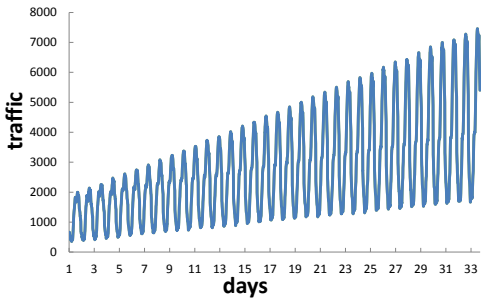

Figure 4-5: Simulated traffic with profile and intensity change

4.2 Estimation

Any given traffic (generated by the simulator itself or not) can be analyzed in the simulator in order to determine its characteristics, such as period or trend, and to produce predictive models for the future behavior of this traffic. This is done in another module, called Estimator, and the given traffic is efficiently saved and sent to this module. To do so, what we call a *Modeled Time Series* (Mdt), is created. This Mdt is formed by three time series which contain for a determined number of hours, each hourly maximum, minimum and average traffic registered.

Once the traffic has been generated, the Estimator module receives the Mdt and is called to produce the models that are later used to make predictions of future traffic, and to be able to make decisions and detect anomalies with this prediction.

The Estimator module has integrated all the algorithms and techniques described in 3.4. It is therefore able to, given a Mdt, obtain for each of the three time series

their period by the *computePeriod* function, compute the standard deviations to different scales of the traffic by the *computeStDev* function, compute the trend by the *computeTrend* function and compute the amplitude by the *computeAmp* function, and obtain a homeostatic series with no trend by the preprocessing performed by these last two functions.

Next step is to produce models for each of these time series. We are going to explain how these predictive models are created from a given Mdt, and how this is later be used to make predictions of the future traffic at any given time. These predictive models are of two kinds: Polynomial models or Neural Network models.

4.2.1 Polynomial model

The polynomial models are based on the polynomial fitting of periodic patterns explained in 3.2. To produce a polynomial model for an OD pair, the Estimator module receives the last updated Mdt of that node. Each of the three time series it contains (maximum, average and minimum) are treated equally. For a given time series, first the period is calculated, and the goal is to produce a profile traffic that has length this period. Ideally, we would want to produce a continuous profile, from these discrete hourly observations, since this will prove to have many advantages when predicting and determining the traffic evolution. In order to give the same importance to all the observations, data is normalized from 0 to 1 by applying a modulus of the calculated period length. To this normalized data points, a polynomial is fitted in the least squares sense, with degree the optimal degree in the BIC sense as explained before.

We observed that by applying this algorithm to obtain a weekly continuous function, the obtained model did fit well the given data, but it had problems at the end of the interval. It had an erratic behavior when it got near the end of the period after the normalization was inverted. This was due to the fact that when applying the normalizing criteria, all points were sent to 0, $1/p$, … $(p\text{-}1)/p$, where p was the period, because of the modulus operation, and so the periodic behavior was not maintained. In order to solve this problem, we were able to check that it sufficed to repeat the points in x=0 to x=1, in order to impose the periodic behavior of the series, and avoid in this way the erratic modeling when the polynomial got close to 1, and equivalently, when the normalized inverted model got close to the end of the period.

Recall that the Mdt saved hourly values for the maximum, minimum and average traffic. This means that the fitting of the polynomial is performed only considering the entire hours, but since a polynomial is continuous, any time between entire hours will have its own maximum, minimum and average prediction. An important observation is that it would not be correct to locate the 3 measures of the Mdt at the end of the given hour, since they affect the whole hour before. To that end, we could either situate the given three measures at time half of the hour where these

minimum, maximum and average were taken, or situate a horizontal segment with that values in the whole hour and fit the polynomial to the stair looking function. We chose to do the first alternative, and from the discrete points situated at half the hour where it was taken, fit the three polynomials.

We have now a continuous model for the periodic profile of the traffic, and it allows us to make predictions of the future hourly maximum, average or minimum traffic by substituting the time to the obtained polynomial.

In order to fit polynomials into a data series in the simulator, the source code polyfitQR, found in [Vi13], which implements the polynomial fitting using QR decomposition in C++ has been integrated into the OMNeT++ simulator.

4.2.2 Neural Model

We next propose a methodology to produce an ANN for a given dataset. We are interested in finding the best neural network structure for the data set we want to fit. To do so, we are going to follow algorithm in Table 4-1. The period p is first calculated, again with the algorithm in Table 3-4, and a first neural network with structure NN(p:1:1) is trained. From it, the number of input lags is reduced one by one, always deleting the input with smallest absolute weight (lines 3 to 5).

Table 4-1: NN Structure Algorithm

1:	Given a time series (*v[i],t[i]), i*=1..*n*
2:	Calculate period p of *(v,t)*
3:	**for** *nlags*=*p*,…,1
4:	Train NN*(nlags*:1:1) and calculate testing error
5:	Eliminate input lag with smallest weight in absolute value
6:	**end**
7:	Choose *nlagsfinal*
8:	**for** *nhidden*= 1..*nlagsfinal*
9:	Train NN(*nlagsfinal*:*nhidden*:1) and calculate testing error
10:	**end**
11:	Choose *nhiddenfinal*
12:	**return** NN(*nlagsfin*al:*nhiddenfinal*:1)

It would be desirable to have small testing errors but also small number of parameters to avoid expensive computational costs. For this reason we propose the following to choose the final number of input lags *nlagsfinal*. We keep for each NN(*nlags*:1:1) the RSS and the maximal error in the test prediction. We calculate the overall RSS minimum, and take the structures that have the RSS under the threshold. We wanted to consider the structures that relatively to the minimal and maximum error are small enough. To do so, we consider that an error is small enough if it is smaller than $minRSS + a(maxRSS - minRSS)$, where $minRSS$ is the

overall RSS minimum, *maxRSS* is the overall maximum, and a is a parameter that can be set by the user. From these structures, we find the minimal maximum error in prediction and also keep the structures that have the maximum error smaller than *minRAXERR+a(maxMAXERR-minMAXERR).* From the structures that remain, we take the one with smaller number of parameters, which means, the smaller number of input lags. With this idea we make sure that we are obtaining the smallest number of inputs that gives a good enough prediction.

When deciding the number of hidden neurons no closed rule exists. As we said before, one hidden layer can approximate any kind of function, and so it should suffice to obtain a good model and avoid computational complexity of a second layer. We therefore only need to decide the number of hidden neurons in this hidden layer. To do this, what is commonly used is to choose the number of hidden neurons between one and the number of inputs. And from this, use cross validation to minimize the test RSS while making sure no over fitting occurs, [Kl09].

When too much hidden neurons are added, error in the training set will decrease but error in the testing set will increase, since over fitting produces bad test predictions. When plotting the RSS, we should notice that the training error decreases when hidden neurons are added (it may arrive to an almost zero in the training set if there are so many parameters that produce a model that is simply a storage of inputs with their output). Nevertheless, so small training errors will usually mean that over fitting is occurring, and therefore will produce bad predictions in the testing set.

Having this into consideration, what we propose to do is the following. We consider all number of hidden neurons from 1 to the number of lags, and calculate its errors in the test set. Again we keep for each NN(*nlags:nhidden*:1) the RSS and the maximal error in the test prediction. We calculate the overall minimum RSS, and take the smallest number of hidden neurons that has RSS smaller than a threshold dependent on the overall minimum and maximum, leading us to the same threshold as before: *minRSS+a(maxRSS-minRSS).*

As for the implementation, we have integrated into the OMNeT++ Simulator the library Open Neural Network Library [Lo14], which implements the multilayer perceptron neural network in C++, with several cost functions, training algorithms and different utilities.

In contrary to what happened with the polynomial model, this model produces the same prediction to the whole hour. This is due to the fact that the Mdt saved hourly values for the maximum, minimum and average traffic. For this reason, the inputs that at a given time receive from the past are the same for the whole hour, and not until the next entire hour, the Mdt is updated.

To solve the given problem when predicting non entire hours we propose the following: linearly interpolate the prediction with the following prediction in order

to obtain a continuous model that approximates better the prediction to the observed traffic.

The given Mdt is the same as for the polynomial model, but it implies another methodology. Again we have Mdt with the last hours maximum, minimum and average traffic. And again this should be placed, in order to center well and make a good interpretation of the model, at half the hour it was saved. When making a prediction, time is normalized in the Mdt saving period, which is one hour. Now two cases arise, depending on the normalized time, observed in Figure 4-6, which shows how the minimum of the Mdt would be situated at half the hours instead of the entire hours.

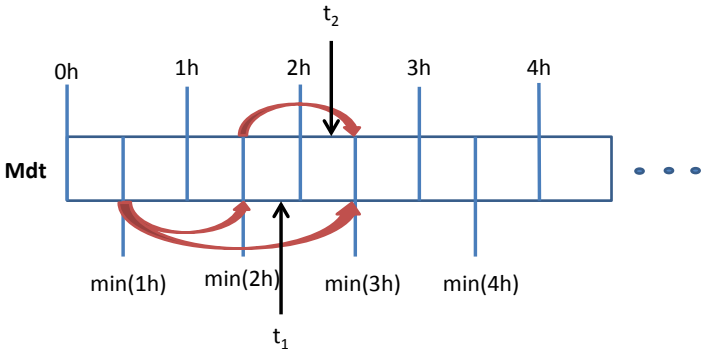

Figure 4-6: Minimum time series of the Mdt

If the normalized time is smaller than one half, meaning that the original time was within the first half hour, Mdt has recently been uptdated, and the last value of the Mdt, *l(Mdt),* is situated less than an hour before (that is the case of t_2 in Figure 4-6). Therefore, we make the next prediction, and interpolate following the next formula:

$$pred(t_{norm}) = l(Mdt) + (NN(Mdts) - l(Mdt))\cdot(t + 0.5) \tag{4.10}$$

If the normalized time is bigger than one half (that is the case of t_1 in Figure 4-6), meaning that the Mdt was updated more than half an hour before, and therefore, the 3 values it contains correspond to more than an hour before, we first make a prediction for next hour, that is actually situated at (t-0.5), that is half an hour before t. We use the hour's maximum, minimum and average prediction to temporarily update the Mdt, that is called Mdt2, so that we can use this hour predictions as inputs, and predict the next hour with the same model. From the next hour prediction and the current hour prediction, join them linearly so that we have a continuous function for any given time. This can be followed in the formula:

$$pred(t_{norm}) = NN(Mdt) + (NN(Mdt2) - NN(Mdt))\cdot(t - 0.5) \tag{4.11}$$

By this methodology, we obtain a piecewise linear function that accurately predicts the future behavior of the time series, as it is seen in Chapter 6.

4.3 Model Validation

After the warmup period is passed, the Analytics module is able to make predictions of the future traffic with the obtained models described before. We are interested in knowing how many of the models are still able to produce accurate predictions as time advances. In order to do so, periodically (at every evaluation period), the Analytics module asks the Evaluator module with the latest version of the Mdt, to compute the chi-squared statistic for each of the average models. That is, compute λ of the formula (3.3), where the expected values e_i are each hours average prediction and o_i the saved average values on the Mdt, with i from 1 to the number of hours saved in the Mdt, *mdtlength,* for the polynomial model or from the biggest input lag of the neural model (in order to have enough past observations to make the prediction), *maxlag*, and to *mdtlength* for the neural model. Therefore, the chi-squared test has *mdtlength-maxlag*-1 degrees of freedom if the model is neural or *mdtlength-1* degrees of freedom if the model is polynomial. The overall algorithm is shown in the following table.

Table 4-2: Fitness_evaluation

1:	Given an Mdt with (*vaver[i],t[i]) i* = 1..*n* and a model *M*
2:	Compute statistic λ (3.3) with
3:	**if** (is.PolynomialModel(*M*))
4:	*b* = *maxlag* +1
5:	*n* = *mdtlength*-1
6:	**else if** (is.NeuralModel(*M*))
7:	*b* = 1
8:	*n* = *mdtlength-maxlag*-1
9:	**end**
10:	**for** *i* = *b…mdtlength*
11:	**if** (is.PolynomialModel (*M*))
12:	e_i = *M(t[i])*
13:	**else if** (is.NeuralModel(*M*))
14:	e_i = *M(vaver[1:i])*
15:	**end**
16:	λ = λ+*(vaver[i]-*e_i*)/*e_i
17:	**end**
18:	Compute p-value of chisqr(*n-1,*λ)
19:	**if** pvalue < 0.05
20:	return VALID
21:	**else return** INVALID

The proposed implementation proceeds as explained, calculating the p-value for the statistic and the given degrees of freedom to determine whether a model is valid or not. In this case we have set a threshold of 0.05, but this can be modified by the user.

4.4 Conclusions

Throughout this section the integration of the traffic generation framework based on OMNet++ has been explained. It consists of two modules that define the basic functions (unitary functions and distribution functions), which are connected to a third module, the function custom, which in turn is connected to the traffic combiner module, finally producing the numerical values for the traffic from an arithmetic equation given by a string and that combines different custom functions.

The traffic generated through this process in the Analytics module is used as input for the network node, thus enabling us to study and evaluate the network performance through simulation, by the creation of data repositories with a condensed representation of the data series to save memory space.

When enough traffic has been generated in the Analytics (the warmup period is finished), this module is call the Estimator module to generate models for the traffic of each pair of OD nodes from the Mdt created for each OD pair.

For each OD pair, three models can be done, one for the maximum, another for the average and another for the minimum time series contained in the Mdt. The three time series are treated in the same way. First given any of these time series, it is preprocessed in order to obtain a homoestecity and stationary in mean series, and the period is calculated, as described in 3.4.2 and 3.4.3. Then the data series would be normalized in time and periodicity would be imposed by adding up data points at *t=1*. The optimal polynomial in the BIC sense would be fitted to the data by minimum squares, obtaining a model of length the previously calculated period. Next, standard deviations for different intensities of the traffic would be calculated in the original series, as described in 3.4.1.

We can now use all of this to predict the future behavior of the time series at a given hour. Once predictions want to be made, the preprocessing of the data series has to be undone, which is done by the following formula:

$$pred(t) = (poly(t) + ct + d)\cdot(at + b) \tag{4.12}$$

where *(a,b)* are the coefficients obtained by the linear regression of the *getHomoestecity* algorithm, *(c,d)* are the coefficients obtained by the linear regression in the *getTrend* algorithm, and poly is the returned value by the polynomial model, which is calculated by:

$$poly(t, p) = \sum_{i=0..d} b_i \cdot \left(\frac{t\%p}{p} \right)^i , \qquad (4.13)$$

where p is the period of the data series, d is the degree of the obtained polynomial and b_i are the coefficients of this polynomial.

As for to perform a prediction by a Neural model, preprocess would not be necessary, since ANN do not require stationarity on the residuals and no further preprocessing other than division by a constant, since the logistic function requires values smaller than 1. Period of the data series would be computed, as explained at 3.4.3. From this period, the optimal structure for the ANN model would be set and trained as explained in 4.2.2. The one-step-ahead prediction of this model would be simply the output of the NN with inputs a subset of the last values observed in the traffic:

$$pred(t) = neural(traffic(t-1),...,traffic(t-p)) \qquad (4.14)$$

Combining either of the predictions obtained by the polynomial or the neural model, with the calculated standard deviations in data, one can obtain a confidence interval of the prediction by achieving a certain confidence in the prediction depending in the value ε.

$$traffic(t) \in (pred(t) - \varepsilon st(t), pred(t) + \varepsilon st(t)) \qquad (4.15)$$

Instead of predicting the maximum, average and minimum prediction, and getting a confidence interval by the standard deviations of the original series, what can also be done, and will result to more accuracy as we see in Chapter 6, is to perform the traffic prediction by the average model, and obtain the interval of confidence by the minimum and the maximum prediction:

$$traffic_{prediction}(t) = pred_{average}(t) \qquad (4.16)$$

$$traffic_{prediction}(t) \in (pred_{\min}(t), pred_{\max}(t)) \qquad (4.17)$$

After the warmup time is passed and models are generated the simulation advances, and at every evaluator period the models are sent to the Evaluator module to check if they are still valid via the algorithm explained in Table 4-2.

This whole process enables us not only to generate accurate models that predict the network traffic, but also gives the Analytics module an intelligent choice on when it is necessary to compute again the models, in case traffic has an evolution and at some point of the simulation some models are not able to predict accurately anymore.

Chapter 5.

Traffic characterization

The aim of this Section is to characterize general traffic properties. To do so, a real dataset is studied and analyzed, in order to compare how different models are able to fit and predict its future behavior, so that later this will be generalized to other traffic samples. To do so, the different characteristics of the dataset are be studied, and exploited in order to find the best combinations of preprocessing and models to fit well the dataset, and accurately predict the future evolution of it. All results in this section were obtained with the statistical software R.

5.1 Descriptive analysis

The dataset considered is a univariate time series which consists of 6 weeks with hourly observations of Internet traffic data from private ISP with centers in 11 European cities [Co12]. Figure 5-1(a) shows the general view evolution of this traffic and Figure 5-1(b) shows a weekly subset of this data.

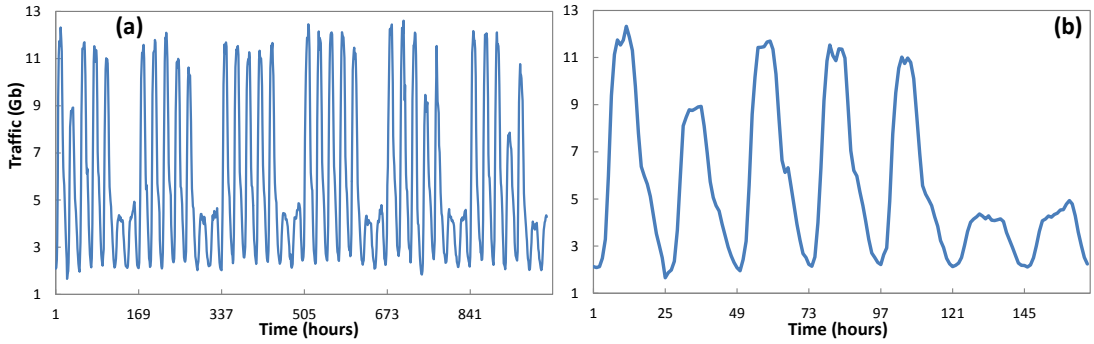

Figure 5-1: Internet traffic: 6 weeks (a) and 1 week (b)

Important properties observed in the traffic and that have been verified by tests on the hourly means and deviations from the whole dataset, are the following:

- In Figure 5-1(a) it can be observed a weekly periodicity in the data, which repeats itself through time.
- In Figure 5-1(b) exactly seven peaks equally separated in a week time length can be observed, which means that there is also a daily period in the data.
- Both figures show an important difference on the intensity of the traffic between the weekly days and the weekend days.
- There are noticeable differences between the weeks, despite the clear daily and weekly profile repetition.
- There is no tendency on the data.

The goal of the different models studied later on in this section is to produce models that fit well the given dataset, keeping its characteristics such as seasonality, correlation and degree of randomness, and also allow us to predict the sixth week's traffic, usually doing one-step-ahead forecasts of each of its hours.

In order to model the traffic and predict the sixth week traffic, the first proposed model, which is the null model, simply takes the hourly means of the first five weeks. The prediction of the sixth week would simply be to calculate the past hourly averages of each previous week. This model would follow the idea of modelling the seasonal component through producing a period length basis traffic like the polynomial model explained in 4.2.1.

Figure 5-2 shows the proposed prediction with the week's average, in blue, versus the real observations of the sixth week, in red. This prediction has a total sum of squared errors of 110.29 and a maximal absolute error of 3.03, and an RSS of 364.70, which implies that, given there are no parameters, the AIC of this model is -700.83.

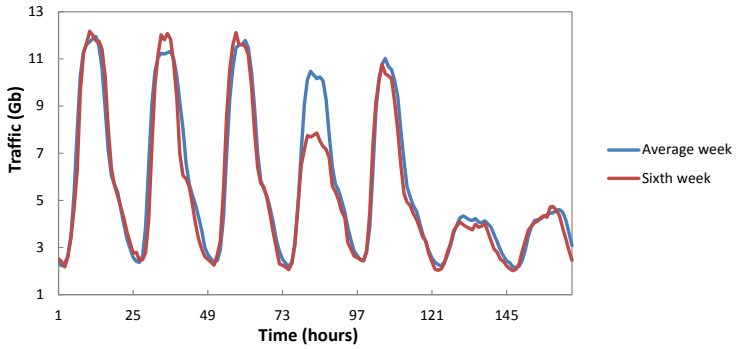

Figure 5-2: Hourly traffic average vs. sixth week traffic

One can observe that this first attempt of prediction shown in Figure 5-2 is much smoother than the real observed series (the random variation is not modeled), and fails to predict well the spikes and each day's maximum value.

Calculating the ACF and PACF of the series, shown in Figure 5-3, one can clearly observe the 24 hour seasonality of the data, and notice that a given observation at time t is highly correlated to past observations, especially to the observations in t-1 and t-2..

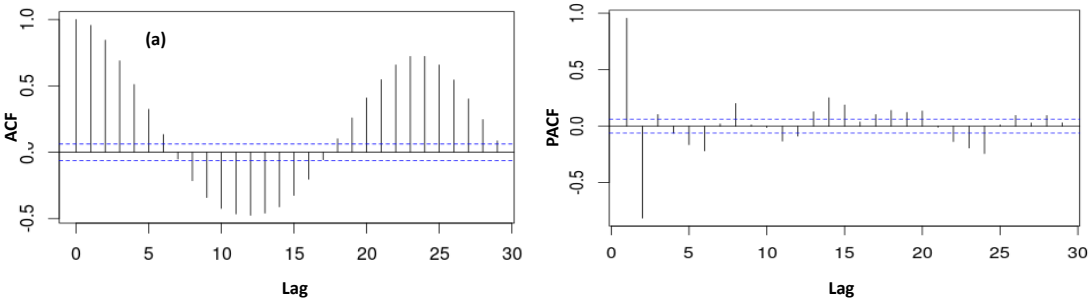

Figure 5-3: Traffic's ACF (a) and traffic's PACF (b)

To get a better fit of the model, it is essential to study the residuals of the time series when subtracting the average week, to see their properties and try to model them. Figure 5-4 shows a plot of the residuals or noise, obtained subtracting to each original week of the dataset the average of the first five weeks. One can notice that, despite the fact that we have subtracted the daily means, there are still clear daily cycles in the new time series obtained, which means that this noise is still not completely random. Notice also the importance of the residuals: it is a series that can explain up to 38% of the value of the traffic in the sixth week (the maximal absolute residual over the observed value).

To have a better understanding of the residuals and determine why they are not completely random, we can have a look at the ACF and PACF of the residuals, shown in Figure 5-5.

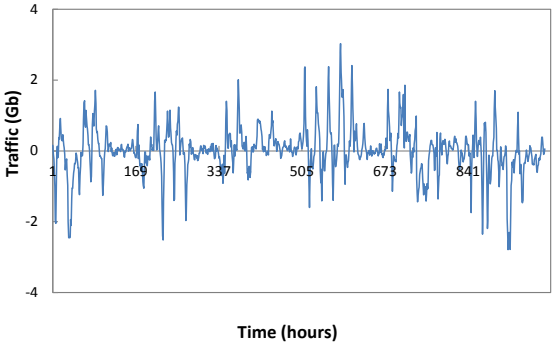

Figure 5-4: Traffic residuals

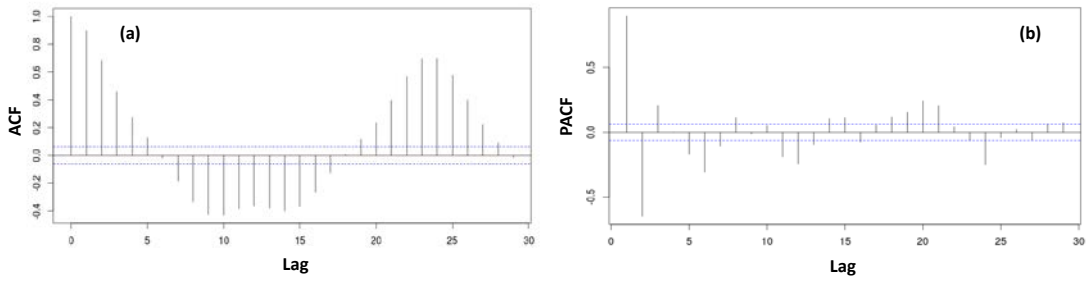

Figure 5-5: Residuals ACF (a) and residuals PACF (b)

The ACF indicates that these 24 hour cycles are due to the correlation given to the past 24 observations, and both figures indicate that the residuals are significantly correlated. Observing Figure 5-4 and Figure 5-5, one can easily notice that the residuals have still a seasonal component, and correlations with past observations are still high.

After the analysis of the traffic and its residuals, the properties observed have indicated us to proceed as follows. We have already discussed in section 3.2 how we propose to obtain the period and model the average profile, automatizing this in order to be able to apply it to other time series. Next, statistical method ARIMA to model the original time series is going to be studied, and also it is used to model this correlated residuals in order to make a more accurate model than the simple average. Finally the approach of time series modelling and prediction with Machine Learning methods will be studied, and their results will be compared to the statistical ones in order to decide which one is more appropriate depending on our goal or the situation we are in.

5.2 ARIMA time series modeling

In subsection 5.1, we observed the correlation in the data and the evidence of non-stationarity, and how important the residuals were to the final traffic. For this reason, an ARIMA model (introduced in 3.1.1) is proposed to try to fit this time series.

Recall that to apply ARIMA to a time-series, data has to be stationary and non-seasonal. Typically to achieve this, preprocessing of data is performed, differentiating to decrease correlations with past observations. This is performed in section 5.2.1. Another alternative would be to try to model the residuals obtained in the previous section as the subtraction of the average traffic to the traffic. This idea is studied in section 5.2.2, and the different methods are later compared.

5.2.1 Original traffic series modelling

The goal of this section is to model the original traffic series with an ARIMA model in the classical way, preprocessing the series by differentiation to obtain a new non-seasonal and stationary series.

To that end, the original series, shown in Figure 5-1(a), is preprocessed in the following order: first the logarithm is applied, in order level the variance, then the new series is differentiated twice, first for a 24 lag and after for a 1 lag, obtaining the non-seasonal and stationary series shown in Figure 5-6.

The ACF and PACF of the series can be observed in Figure 5-7, showing that the series is now much less correlated than it was before. Observe though, that the correlation to the first and to the 24th lag is still significant, but further differentiation does not improve this. This will mean that the usual techniques to chooses the coefficients (p,q) for the AR and the MA from the PACF and the ACF will not be able to be applied

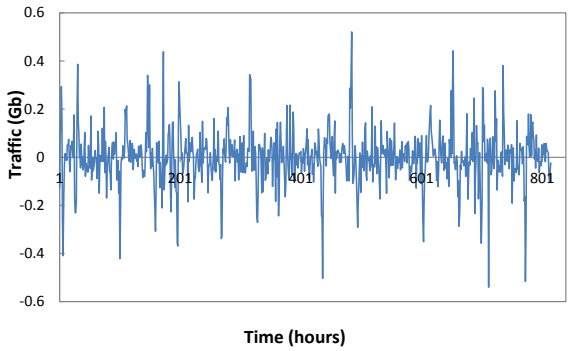

Figure 5-6: Logarithm of the series differentiated

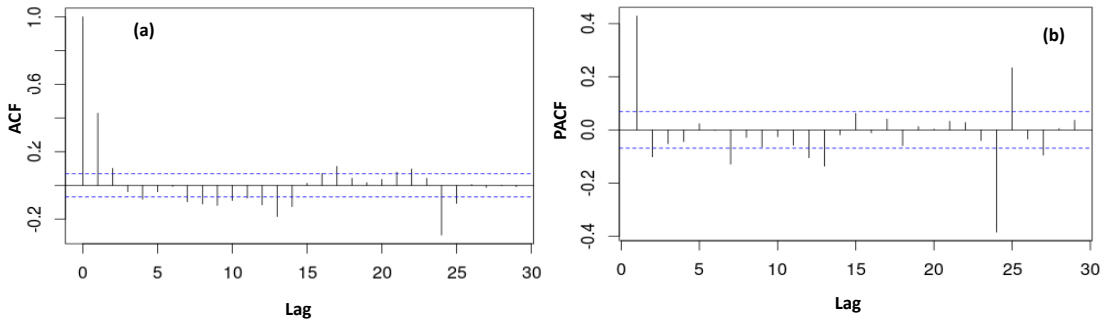

Figure 5-7: ACF and PACF of logarithm of the series differentiated

We propose to make the decision based on finding the coefficients (p,d,q) that minimize the AIC (which is the way R functions like auto.arima work). Given that the series is no longer seasonal, we are only going to consider models of the type (p,0,q), and since other than the 24th lag, the significant correlations are in the first three lags, we consider p and q from 0 to 4.

It can be observed in Figure 5-8(a), which shows the AIC for the ARIMA models (p,0,0) and (0,0,q), that the minimal (p,0,0) AIC is achieved for (2,0,0) and the minimal (0,0,q) is achieved at (0,0,2). It is necessary now to consider also the ARIMA models mixing the AR and the MA. With the observed facts before, it is intuitive to think that the minimal AIC will be around the ARIMA(2,0,2) model. Figure 5-8(b) shows the AIC for models (1,0,q), (2,0,q) and (3,0,q), which achieve the minimal at ARIMA(2,0,1), with an AIC of -1473.54 (very close to ARIMA(2,0,2), as predicted, which has an AIC of -1473.50).

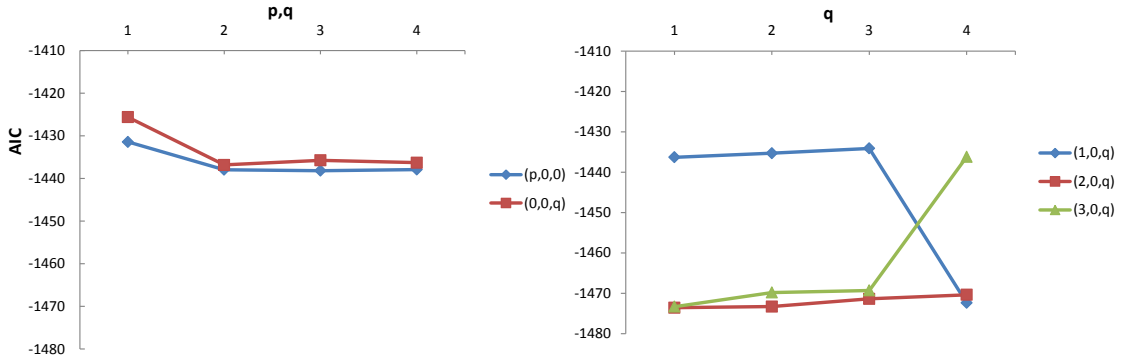

Figure 5-8: (a)AIC of ARIMA(p,0,0) and (0,0,q) models and (b) (1,0,q),(2,0,q) and (3,0,q) models

It is time to fit the optimal ARIMA(2,0,1) model in the AIC sense to the data. Obtaining the following model for the differentiated residuals:

$$X(t) = 1.39X(t-1) - 0.461X(t-2) + \mu + e(t) - 0.9e(t-1) \qquad (5.1)$$

Since $E(X) = 0$, this implies $\mu = 0$. And MA(1) satisfies $e \approx N(0, \sigma^2)$, with $\text{var}(X) = \sigma^2(1 + 0.9^2)$, since var*(xlogdiff)= 0.012*, this implies that $e \approx N(0, 0.007^2)$. Preprocessing needs to be inverted in order to obtain the final model for the traffic, this means differentiation needs to be inverted and the exponential applied to the resulting series.

Figure 5-9 shows the prediction of the last week with this model versus the observed values. This model has an AIC of -1473.54, a total squared prediction error of 42.72, and a maximal absolute error of 2.62. Notice how compared to the null model, this model is able to predict better the spikes, maximums and minimums.

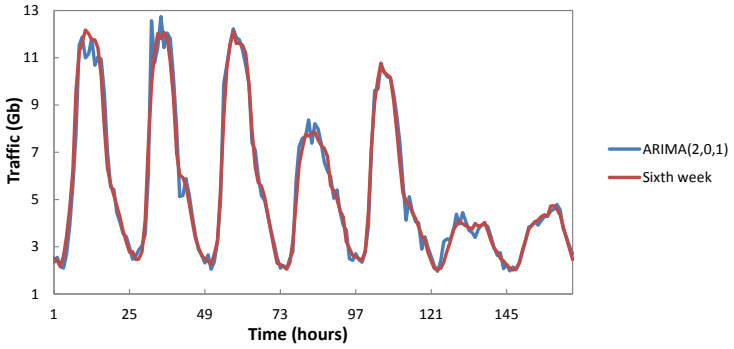

Figure 5-9: : Arima(2,0,1)'s prediction vs observed values

5.2.2 Residual traffic series modelling

In this section, traffic is modeled as a sum of the average mean traffic and the residuals. Residuals are not be modeled as a normal distribution, but by an ARIMA model, to be able to have the correlations amongst them into consideration.

To achieve this, first the data residuals are obtained subtracting the week's hourly average. A plot of these residuals was shown in Figure 5-4. By observing Figure 5-5, we notice that there are still 24 hour cycles, and the ACF and PACF on the new series indicate high correlation to lag *(t-1)* too, so we differentiate it twice: first for a 24 lag and after for a 1 lag, obtaining the desired stationary and non-seasonal series, shown in Figure 5-10. We apply the R function auto.arima, which compares different ARIMA(p,d,q) and return the one that fits the best, in this case (1,0,2).

```
Series: xlogdef112
ARIMA(1,0,2) with zero mean

Coefficients:
         ar1      ma1      ma2
      0.7654  -0.7202  -0.2592
s.e.  0.0277   0.0434   0.0425

sigma^2 estimated as 0.1649:  log likelihood=-421.94
AIC=851.89   AICc=851.94   BIC=870.7
```

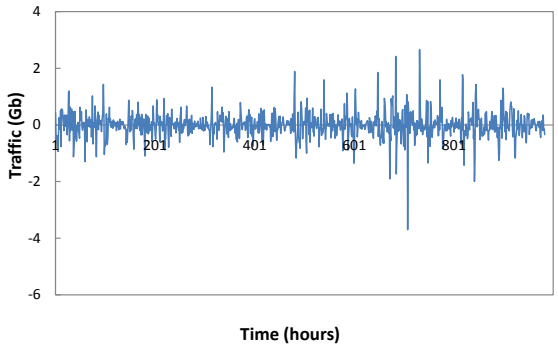

Figure 5-10: Traffic with the hourly average subtracted and differentiated

To check the correctness of the result, some checks are carried with the residuals: the ACF and PACF contain no significant correlations and randomness is verified by the Box-Pierce test. Normality is not verified by the Shapiro-Wilk normality test, and normal skewness and kurtosis is also not verified with the Jarque Vera test.

```
        Box-Pierce test

data:  res
X-squared = 0.0887, df = 1, p-value = 0.7659
```

```
        Shapiro-Wilk normality test

data:  res
W = 0.9426, p-value < 2.2e-16
```

```
        Jarque Bera Test

data:  res
X-squared = 1430.878, df = 2, p-value < 2.2e-16
```

Figure 5-11 shows the histogram of the residuals, and it's QQ-plot. It can be observed that the residuals have a normal profile, but the spike is too high for a normal distributed series.

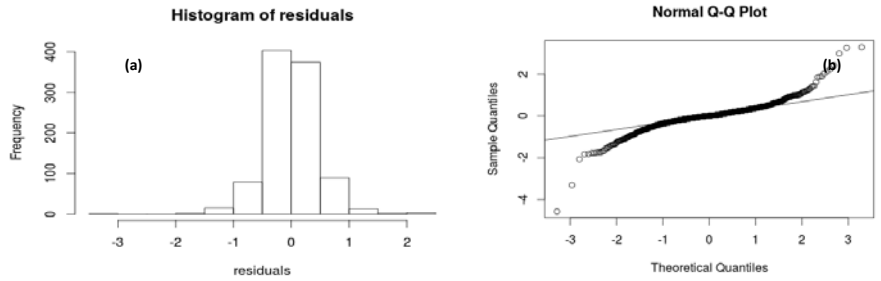

Figure 5-11: ARIMA's residuals histogram (a) and QQ-plot (b)

Applying the obtained ARIMA(1,0,2) to predict the sixth week of the time-series, one hour at a time, and inverting pre-processing of the data, a prediction of this sixth week can be performed. Figure 5-12 shows a plot of this prediction, in blue, against the real observed values of last week, in red. This model produces an AIC of 851.89.The squared error of the prediction is 78.12, and the maximal error 2.82. It satisfies the following formula (preprocessing will have to be inverted to obtain the prediction):

$$X(t) = 0.76X(t-1) + \mu + e(t) - 0.72e(t-1) - 0.26e(t-2) \qquad (5.2)$$

Since $\mathrm{E}(X) = 0$, this implies $\mu = 0$. And MA(2) satisfies $e \approx N(0, \sigma^2)$, with $\mathrm{var}(X) = \sigma^2(1 + 0.72^2 + 0.26^2)$, which implies that $e \approx N(0, 0.3^2)$.

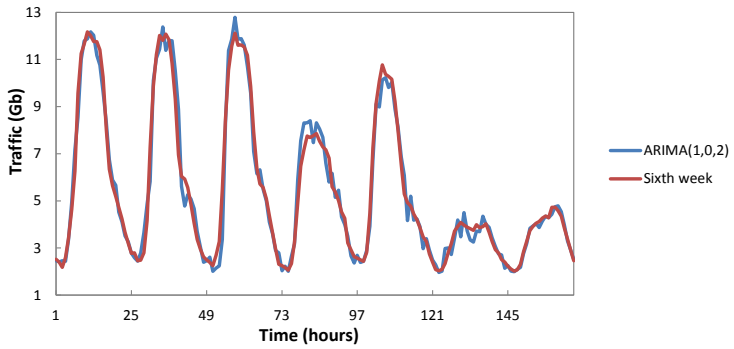

Figure 5-12: Arima(1,0,2)'s prediction vs observed values

The minimal AR model in the AIC sense is the obtained by an ARIMA(4,0,0), as can be observed in Figure 5-13, which shows the AIC for different ARIMA(p,0,0) models. It has an AIC of 871.31, a predictive squared error of 80.29 and a maximal prediction error of 2.82. The resulting model for the preprocessed series is:

$$X(t) = 0.08X(t-1) - 0.2X(t-2) - 0.04X(t-3) - 0.19X(t-4) + e(t) \qquad (5.3)$$

With $e \approx N(0,0.47^2)$.

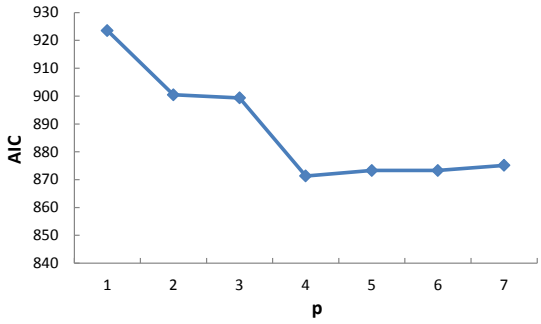

Figure 5-13: AIC plot of ARIMA(p,0,0), varying p

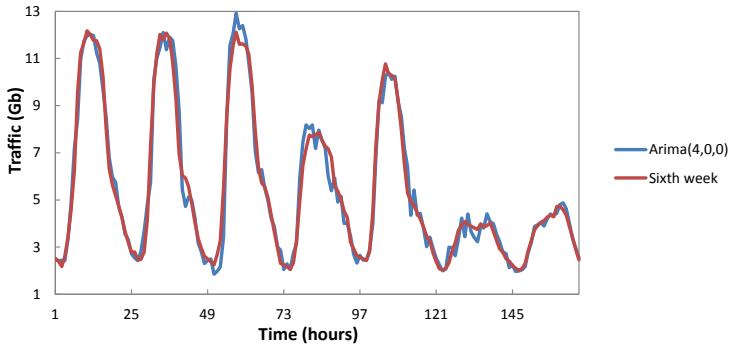

Figure 5-14: Arima(4,0,0)'s prediction vs observed values

The final prediction is computed by the obtained ARIMA model, and reverting the pre-processing of the data. First, traffic at time t is expressed as the sum of the hourly mean (described in the previous subsection) and a first noise that we call *res*: $traffic(t) = average(t) + res(t)$. The series *res* was differentiated twice, first for a 24 lag, meaning that we obtain a new series $res'(t) = res(t) - res(t-24)$, and afterwards for a 1 lag, meaning that another time series is obtained $res''(t) = res'(t) - res'(t-1)$. It is to this final time series *res"*, that an ARIMA model is fitted to model, and therefore to the *res"* obtained as a prediction in ARIMA, all the process has to be inverted to obtain the prediction of traffic.

5.3 Neural Network time series modelling

In this Section we are going to follow the same structure as for the ARIMA time series modelling section, to propose two different ways of computing Neural Networks models to model and predict our original traffic data series.

5.3.1 Original traffic series modelling

The first attempt to model the original time series is to consider the original time series with no preprocessing, and therefore, this could be performed knowing very little on the properties of the time series.

To fit a Neural Network to our data series that has two seasonal components, a daily and a weekly period, one could try to fit a NN with inputs the whole week before the given observation, but given that our data set only has 5 weeks of training data, this would not allow training the NN. Also, this would imply a huge number of parameters, over 2000 parameters if we consider 12 hidden neurons, so this is not viable.

We will see that in this case, to make a good model and prediction, it suffices to give as inputs for a prediction in time t the past 24 observations, meaning that we are only using the fact that there is a daily period. We implement a NN(24:12:1), a Neural Network with 24 inputs, 12 neurons in the hidden layer, and 1 output. The resulting prediction of the sixth week versus the sixth week's original data can be observed in Figure 5-15.

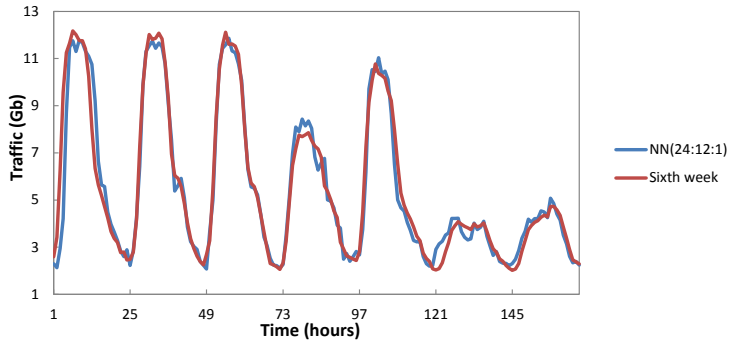

Figure 5-15: NN(24:12:1)'s prediction vs observed values

This prediction has a squared error of 21.05 in respect to the observed values, a maximal prediction error of 1.40, and the model has an AIC of 626.39 and a BIC of 2098.87. It has an obvious problem, though: the number of parameters (and consequently the BIC and AIC) is huge. We are talking about 302 parameters.

Neural Networks are non-deterministic, which means that there can be many optimal models for a problem. For this reason it may not be wise to try to make sense out of the coefficients of a solution: very small weights in an input might not mean that this input is not important, since it might have greater weight in another optimal model. Nevertheless, it has been shown [Gu03] that inputs that are less correlated to the final output usually have smaller weights in all the optimal models. This is why we propose to reduce the number of inputs, by selecting the ones that are more correlated to the output, (and consequently should have smaller weights in the NN(24:12:1) and will give less significant difference on the output if deleted).

To contrast the relationship between the importance of the weights for each input lag and the obtained importance through the ACF and PACF, we have computed a NN(24:1:1). Each input weight has now a direct impact to the hidden neuron and by it to the output, so it should indicate the importance of this lag in respect to the rest. Table 5-1, shows the weights obtained in this model.

PACF ordered the input lags importance in the following order: 1, 2, 6, 24, 14, 15, 8, 19, 20, 17, 18, 5, 11, 23, 13, 3, 4, 22, 10, 12, 16, 21, 7, 9. There are certainly similarities with the importance of the weights obtained by the NN(24:1:1) and the one obtained by the study of the correlations in the data: lag 1 and 2 have, with difference, the biggest weights, and were also, with difference, the most correlated to the observation at time *t*, lags 6 and 24 have in relation to the rest of the weights also big weights, but are not in this case the third and fourth as in the correlation importance, but seventh and fourth respectively. In contrast, lags 7 and 9, that were the least correlated, have now the third and tenth biggest weights. So it can be asserted that even if there is some relationship (with lags 1 and 2 it is very clear), the relationship is fuzzy.

Table 5-1: NN(24:1:1).

Lag	Weight
Intercept.to.1layhid1	1.34796675
Lag1.to.1layhid1	5.3445143
Lag2.to.1layhid1	2.46391075
Lag3.to.1layhid1	0.01552581
Lag4.to.1layhid1	0.12927058
Lag5.to.1layhid1	0.1061852
Lag6.to.1layhid1	0.45473349
Lag7.to.1layhid1	-0.7807904
Lag8.to.1layhid1	0.20208978
Lag9.to.1layhid1	0.34824336
Lag10.to.1layhid1	0.36010691
Lag11.to.1layhid1	0.34216074
Lag12.to.1layhid1	0.37881046
Lag13.to.1layhid1	0.08025356
Lag14.to.1layhid1	0.03977197
Lag15.to.1layhid1	0.51267712
Lag16.to.1layhid1	0.44479635
Lag17.to.1layhid1	0.12961115
Lag18.to.1layhid1	0.08695083
Lag19.to.1layhid1	0.04012604
Lag20.to.1layhid1	0.07237458
Lag21.to.1layhid1	0.10325807
Lag22.to.1layhid1	0.17865408
Lag23.to.1layhid1	0.55288885
Lag24.to.1layhid1	0.68154038
Intercept.to.Lag0	-0.2382322
1layhid.1.to.Lag0	1.37555873

The model NN(24:1:1) has and AIC of 54.91, a squared prediction error of 26.39 and a maximal error of 1.92.

Table 5-2, with the weights of the trained NN(4:1:1) is going to be used to show how the weight criteria to eliminate less important lags is not deterministic.

Table 5-2: NN(24:1:1).

Intercept.to.1layhid1	0.82080497
Lag1.to.1layhid1	-4.3045656
Lag2.to.1layhid1	1.96199814
Lag6.to.1layhid1	0.08686821
Lag24.to.1layhid1	0.01492473
Intercept.to.Lag0	1.24540968
1layhid.1.to.Lag0	1.74005087

If we compare Table 5-1 and Table 5-2, we can notice that there is not a direct relationship between the importance of the weights order: in the first case we would have ordered by 1,2,24,6, while in the second case the order of importance would be 2,1,6,24. This is why one needs to be careful when using this criteria to reduce from 24 to 1 the number of inputs, and other criteria, such as always deleting the less correlated lags, studying the PACF of the series, might be more appropriate.

We are interested in finding the number of inputs that gives a better combination of accuracy and parsimony. We have noticed that AIC similar to linearly increases with the number of inputs when the number of hidden neurons is big. For all models, with hidden neurons from 2 to 24, the minimal AIC is achieved for n=1 inputs. That is not the case when the number of hidden neurons is just one, where the minimal AIC is achieved with n=2 inputs, which is actually the overall minimal AIC of all the models. Figure 5-16 shows the evolution of the AIC while increasing the number of inputs for models with 12, 3, 2 and 1 hidden neurons. It is also interesting to consider the evolution of the BIC of the model, shown in Figure 5-17. Finally the relationship of the predictive squared errors and the number of hidden and input layers is shown in Figure 5-18.

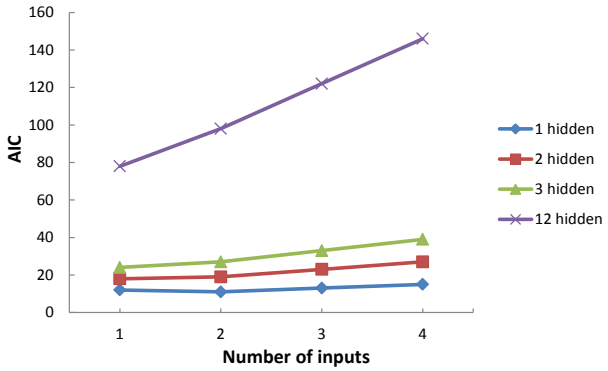

Figure 5-16: AICs for different Neural Network models

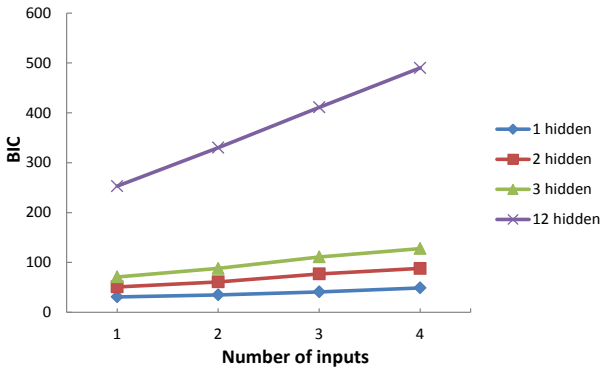

Figure 5-17: BICs for different Neural Network models

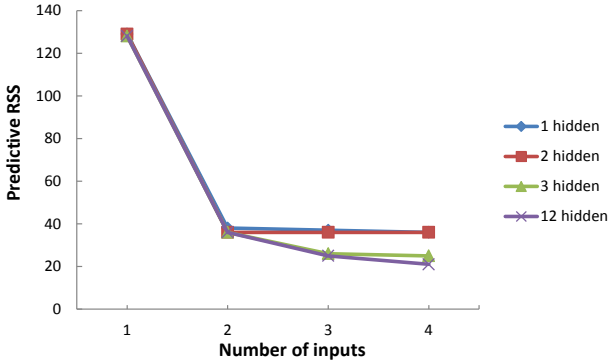

Figure 5-18: Predictive RSS for different Neural Network models

Obviously the error increases when the number of input decreases: less input information is given to compute the output. But there is an important difference in the values from 1 input (error around 128), to errors produced with 2, 3, or 4 inputs (errors from 21.37 to 37.89). So it can be asserted that, while 2 or more inputs suffice to define well the NN model, 1 input does not suffice. That is why, even if the number of parameters is much smaller, the minimal AIC is not achieved in any of the NN with only one input, and is achieved instead in the NN(2:1:1).

Another thing that can be observed from Figure 5-18, is that the error usually increases also if the number of hidden neurons decreases. This is a sign that there was not overfitting in the NN with 12 hidden layers, and allowing less parameters in the NN makes the NN to have a bigger error afterwards.

Nevertheless, if we consider the evolution of these models' BIC, shown in Figure 5-17, we find that the minimum BIC is achieved in the NN(1:1:1) model, even if the error for the models with only one input is much bigger than the other models. This is because the BIC puts a bigger disadvantage weight on the number of parameters than the AIC.

This is a clear example were, choosing the model as the optimal in the BIC sense would lead us to choose a bad model, and that's why it is important to check other means of error. In this sense, some when modelling a NN, some part of the data is saved for cross-validation and testing, and it is to the testing data that one can calculate the errors of prediction in the same way we did in Figure 5-18.

We now train the optimal NN(2:1:1) model in the AIC sense (with AIC=11.33), which also lead to a small enough squared error in the sixth week's prediction (testing data of our data set). Training this network with the first five weeks the following predictive model for the time series (preprocessed in order to normalize it, dividing by 13) is obtained in the following formula.

$$X(t) = f(1.58 f(4.89 X(t-1) - 2.36 X(t-2) - 1.07) - 0.35 \quad (5.4)$$

where f is the logistic function $f(x) = \frac{1}{1+e^{-x}}$

Figure 5-19 shows how the prediction of the sixth week looks like. This prediction has a squared error of 36.94 and a maximal absolute error of 1.96, which is worse than the 21.05 of the initial NN(24:12:1), but has just 5 parameters in contrast to the 302 of the other model (and that is why the AIC is much smaller). To better visualize this model, Figure 5-20 shows a representation of this Neural Network.

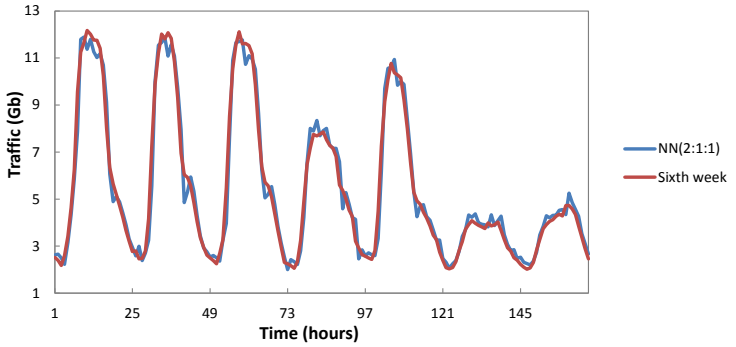

Figure 5-19: NN(2:1:1)'s prediction vs observed values

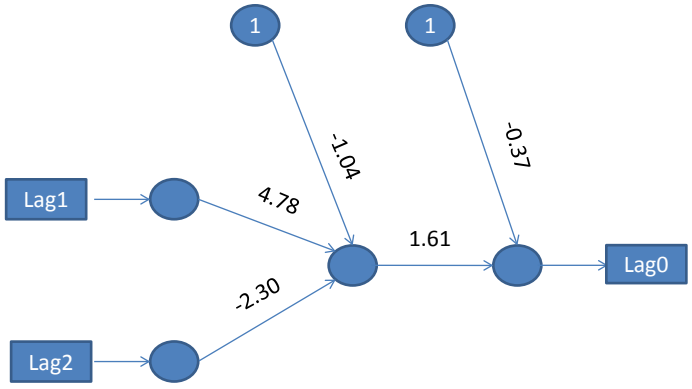

Figure 5-20: NN(2:1:1)

5.3.2 Residual traffic series modeling

Following the two preprocessing methods in which we applied the ARIMA models, we are also going to try to model the residual traffic with Neural Network methods: the data series will also have its weekly mean subtracted, and to this new time series the Neural Network model will be fitted.

Therefore we are now going to consider the residual traffic, that was shown in Figure 5-4. Recall that this traffic had still a daily period, and because of this the first Neural Network that will be trained will have as inputs the 24 past observations. We are going to consider, as before, a NN(24:12:1), and train it, this time, with the residual traffic series. To the obtained predictions, the weekly hourly average will be summed up. The resulting model has an AIC of 627.74, and the squared sum of predictive errors in the sixth week is 49.59, with a maximal absolute error of 2.08. Figure 5-21 shows the plot of the prediction of the sixth week versus the observed values; in this case, we notice at simple sight how the prediction does not fit as accurately as the past models the last week.

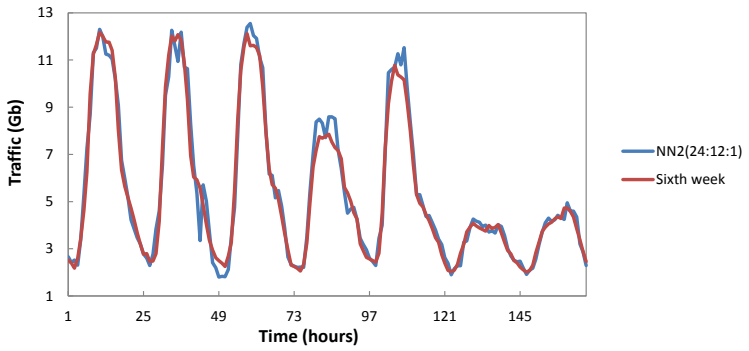

Figure 5-21: NN(24:12:1)'s prediction vs observed values

Considering now the NN(24:1:1) model, we obtain a model with AIC of 58.08, and a squared sum of predictive errors in the sixth week of 20.29, with a maximal absolute error of 1.23. Notice that reducing the number of hidden neurons lead to a smaller predictive error, which means that there is a clear overfitting in the first model, NN(24:12:1), due to the big number of parameters that are trained. We proceed again to look for the best NN model in the AIC sense.

The evolution of the AIC when varying the number of inputs and the number of hidden neurons, which is shown in Figure 5-16, is similar to the evolution in the original traffic series models, but now the minimal AIC is achieved in the model NN(1:1:1), with only one input *(t-1)* and one hidden neuron. This model has an AIC of 13.44, a squared sum of errors in the prediction of the sixth week of 26.9 and a maximal absolute predictive error of 1.41. It's predictive performance is shown Figure 5-23, which plots the sixth week prediction of the NN(1:1:1) versus the observed values

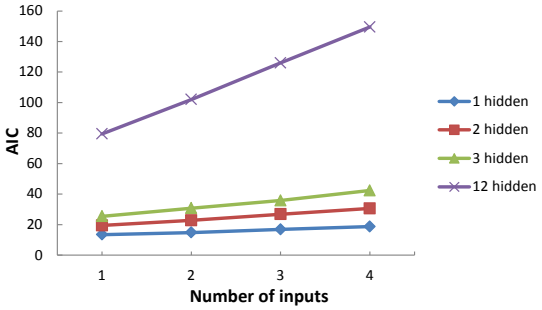

Figure 5-22: AICs for different Neural Network models

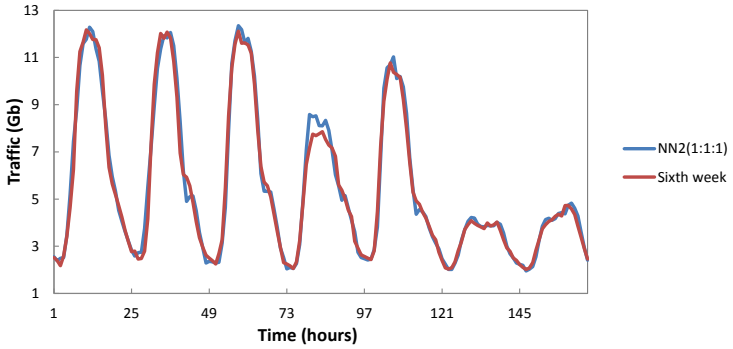

Figure 5-23: NN(1:1:1)'s prediction vs observed values

It can be observed that the predictive results are better than the model NN(24:12:1), due to the overfitting of this model, but are slightly worse than the NN(24:1:1). This is because the AIC disadvantages models with higher number of variables (this model has only 4 parameters, while the other had 15 parameters, and the error between them is not that different).

Notice also that in this case, where data had its mean subtracted, the error for NN(1:1:1) is 26.9, while we saw in the last section that the NN(1:1:1) applied to the original time series (no average subtraction) had a much bigger error, of 129. This can be explained as follows: in this case, we are only modeling the noise, and because of this, 1 input suffices to give the information required to fit well the model. On the other hand, 1 input did not suffice when modeling the whole original time series with no mean subtraction, since more information was held in this time series.

The obtained NN(1:1:1) model can be visualized as a Neural Network in Figure 5-24, and follows the following equation, with activation function the logistic function:

$$X(t) = f(1.71 f(2.14 X(t-1) + 0.07) - 0.88 \quad (5.5)$$

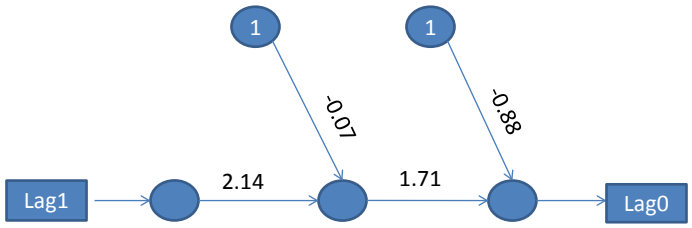

Figure 5-24: NN(1:1:1)

Since this model is applied to the preprocessed data which had the weekly mean subtracted, and was divided by 4 in order to normalize the data to be able to train the model with the logistic function, the final model can be written as:

$$\text{traffic(t)} = 4X(t) + average(t) \tag{5.6}$$

$$= 4(f(1.71 f(2.14 res(t-1) + 0.07) - 0.88) + average(t) \tag{5.7}$$

$$\text{and } average(t) = weekprofile(t\%168)$$

5.4 Conclusions

In this Section a real data series has been studied in order to be able to exploit the analyzed properties and produce from them different models that represent well the generation process and can predict accurately future observations.

In subsection 5.1, the time series has been analyzed, detecting its periods, and the relationship between the weekly average and the resulting residual series obtained when subtracting this average.

Due to the importance of the seasonal traffic profile, the previous chapters have been devoted to the production of a continuous profile form the discrete past observations. But modeling the average traffic was not enough to model the time series, since the residuals still have an important degree of explanation of the final traffic. For this reason, in subsections 5.2 and 5.3 we have modelled the original traffic data from two points of view: considering the original traffic data as a whole, and considering the original traffic data as a sum of a traffic average plus the residuals. Both approaches of the problem have been dealt with Statistical methods first, fitting some ARIMA(p,d,q) models into the data series, and with Machine Learning methods afterwards, training different Neural Network models with the data series. We proceed now to make a comparison of the obtained models and their performances.

We are first going to compare the models that consider the original data set as a whole, with no subtraction of the weekly average. Since we are considering very different models some properties of the models need to be taken into consideration in order to make a fair comparison of the models.

The number of parameters is obviously important to determine the complexity of each model. Bigger number of parameters usually implies better accuracy if overfitting is not occurring, but it is also a drawback for the computational cost of computing them, and the interpretability of the final model. There is a problem when comparing the number of parameters of ARIMA models with NN models, though. Recall that the best ARIMA model was ARIMA(2,0,1). This model has, 3 parameters, but to achieve it data needs to be preprocessed applying a logarithm and differentiating it twice. On the other hand, the optimal Neural Network model was NN(2:1:1), with 5 parameters, and data was also preprocessed, but just normalizing it dividing it by 13.

Seeing this, we need to realize that even if the number of parameters is smaller for the ARIMA models, preprocessing was much complex, and therefore, truly fair comparison on complexity cannot be made comparing only the AIC.

Also the accuracy when fitting the training data is important for determining how good a model is. For this we consider the AIC, since it takes into account the errors when fitting a model and puts a disadvantage with the increase on the number of characters. Again, AIC can be used to compare ARIMA models between them, and NN models between them, but it is not completely safe to compare the AIC from a NN to an ARIMA, because of the not considered complexity due, for example, with the preprocessing, and also because the residual errors are measured on a different scale (recall that to apply ARIMA logarithm and differentiation was applied, while NN were normalized).

Having this in mind, to make a complete fair comparison of all the models, one should measure the predictive accuracy in respect to the real observed values, that is saved for data for testing, which is what we did with the sixth week of our data set. For this reason, we are going to measure each hourly error of the sixth week prediction for each model and sum the squared of all of them. This will also allow us, later on, to compare these models with the models predicting the residual data series, since the prediction will be comparable after summing again the weekly average. This predictive performance will be measured with the squared sum of errors RSS, and the maximal absolute error in the prediction.

Table 5-3 shows the different number of parameters, AICs, predictive RSS and maximal error in prediction for the models considered when modelling the original data series as a whole. The same is done, in Table 5-4, for the models that model the residual series, and sum the obtained outputs to the weekly average afterwards.

Table 5-3:Original traffic series modelling performances.

Model	Number of parameters	AIC	Predictive RSS	Maximal error in prediction
ARIMA(2,0,1)	3	-1473.54	42.72	2.62
NN(24:12:1)	302	626-39	21.05	1.40
NN(24:1:1)	27	54.91	26.39	1.92
NN(2:1:1)	5	11.33	36.94	1.96
NN(1:1:1)	4	11.54	129.1	2.4

Table 5-4:Residual traffic series modelling performances.

Model	Number of parameters	AIC	Predictive RSS	Maximal error in prediction
Null Model	0	-700.83	364.70	3.03
ARIMA(1,0,2)	3	851.89	78.12	2.82
NN(24:12:1)	302	627.74	49.59	2.08
NN(24:1:1)	27	58.08	20.29	1.23
NN(1:1.1)	1	13.44	26.9	1.41

The first conclusion that derives from the null model, taking the hourly average as the model, is that even if the average fits quite well the profile of the sixth week, it is not enough in the sense that the residuals have an important significance in the final traffic: they can explain up to 48.1% in the sixth week traffic.

The first attempt to model the whole traffic with an ARIMA model gives an optimal ARIMA(2,0,1) in the AIC sense. With the differentiation and the dependence on the *x(t-1)* and *x(t-2)* we are able to produce a much more accurate model than the null model. Also, compared to the ARIMA models that model the residuals and not the data series as a whole, it has the minimal AIC and minimal predictive RSS. It can be asserted, because of this, that it is not worth it to subtract the average in the case of the ARIMA models when we want to predict traffic, since it gives a worse fitting and predictive performance.

In the case of modeling with Neural Networks we have observed the following. Even if there is a weekly period, trying to give as input a whole week before or even some observations of the past week would not allow to train the NN since we only have 5 training weeks, and it would need a huge number of parameters, over 2000 if we consider 12 hidden neurons. Nevertheless, this is not a problem, as we have

proved that taking as inputs the past 24 observations suffices to train curate model.

In order to reduce appropriately the number of inputs we have studied the relationship between the importantly correlated lags, and the lags that have the biggest weights in NN with one hidden neuron, NN(24:1:1). It can be asserted that there is some relationship, which in the case of the two more correlated lags is obvious, but this relationship is weaker in other cases such as the less correlated lags. Moreover, considering a NN with the 4 more important lags of the original NN(24:1:1), NN(4.1:1) and training it produces another order of importance of these four lags. This proves the non-determinism of the method of choosing importance of a lag with the weights of a NN, which adds up to the non-determinism of the final optimal models (which can produce different optimal models with different weights) problem, and has made us choose to determine this importance through the importance on the correlations shown in the PACF, and not through the importance on the weights of the inputs of NN(24:1:1).

We also observed that for the NN models that take the original time series as training data, the minimal predictive error is achieved in the biggest NN(24:12:1). This indicates that the 24 lags had some importance for predicting the output, and eliminating them makes the errors in prediction increase, and also that no overfitting occurred.

When modeling the data series as a whole, the minimal AIC for the NN models was achieved at NN(2:1:1). Based on the evolution of the AIC for the different NN models, we could observe that for models with hidden neurons from 2 to 24, the minimal AIC was achieved at n=1 inputs, except when there is only one hidden neuron, that has a minimal AIC in n= 2 inputs, which is the overall minimum. This can be explained studying the evolution of the predictive RSS for these models: error increases when the number of inputs and hidden neurons decrease, but there is a very big difference between the error of models with n=1 inputs and the rest. This causes that even if there are less parameters in N(1:1:1) than in NN(2:1:1), the AIC is worse because the error is much bigger. Therefore, it can be asserted that when modeling this data series, n=2 inputs suffice to train a quite accurate model, while just 1 input does not: lag 2 explains an important percentage of the final output and deleting it produces a much bigger error.

Also the evolution of BIC has been studied for these models, and a worrying conclusion has been raised: in this case the minimal BIC is achieved in NN(1:1:1). Seeing how the predictive error of models with just one input are around 128 while all the other models have errors from 21 to 38, choosing NN(1:1:1) would be a very bad choice, even if the BIC is minimal. This happens because the disadvantage in the number of inputs the BIC sets is bigger than in the AIC, and produces this result. Therefore, one has to be careful when choosing a model, and always do calculate the errors on the testing set in order to avoid this kind of problems.

The minimal predictive error for the NN (and for all the models, actually), is achieved at the NN(24:1:1) that takes as input data the residuals. Recall that modeling the residuals in the ARIMA models produced worse predictions, but in the NN models the contrary happens. The predictive errors are much smaller, even for NN(1:1:1) the error is only 26.9 compared to the 129 of NN(1:1:1) when modeling the whole data set. This means that after subtracting the weekly mean, one input suffices to explain the resulting data series, while without subtracting the average 2 inputs were the minimal number of inputs to explain well the data series.

Notice also that the error for the biggest NN(24:12:1) was worse than the errors of smaller NN models. This means that overfitting occurred, and too many parameters were being trained for this model. Again this did not happen in the case of NN(24:12:1) with the original time series, and it has the same explanation as the error of NN(1:1:1): the complexity of the series after subtracting the mean decreased so much, that NN(24:12:1) produces overfitting and just one input suffices to explain well the data series.

In the following sections, we want a model that allows us to generate different profiles of traffic. In this case, we had a clearly work oriented traffic, with maximums during the morning, and small traffic on the weekends and during the night. We would like a model that, other than giving good predictions, it allows us to adapt the generation of traffic easily and intuitively to other situations as well. For example, we could be interested in modeling a more leisure oriented traffic, where the maximums would be during the afternoon and on weekends, or we might need to add more correlations, a tendency or increase randomness.

The NN model gives us a very accurate prediction with a small number of parameters and with a very simple preprocessing of data (dividing by the maximum suffices). Therefore, it is the method that we will use in the future when we need to predict a given traffic. A possible approximation would be to calculate the period traffic with the proposed algorithm in subsection 3.2, and train a NN with inputs all lags within the period. Afterwards the number of inputs could be reduced as long as the error doesn't increase more than a given threshold from the initial error produced. This will be studied in the following sections.

Nevertheless, if our goal is to generate traffic, Neural Networks have an obvious problem: it is very difficult to interpret how each coefficient affects the final output, since, as it can be observed in the formula, it has the activation function which gives a counterintuitive result. It would be hard to decide how to modify the coefficients in order to adapt the traffic generation to another profile, say with higher maximums or with another seasonal component, or with a desired correlation of the generated data.

On the other hand, the ARIMA model works completely different. Maybe the prediction is not as accurate, but there is an hourly average that acts as a basis for the traffic, the coefficients indicate exactly the correlations with the two

observations before, and the final coefficient acts to add some random noise to the prediction. It is, in this case, very intuitive how to modify the average base profile to adapt to other traffic profiles, how to modify the coefficients to add or extract correlation to the previous observations, and how to increase or decrease the randomness of the traffic generated by changing the distribution function of the noise. The fact that the construction and coefficients of the model are so intuitive and easily interpretable gives us flexibility to generate different traffic profiles satisfying the desired properties that each situation requires.

To conclude, different models have been studied in this section for our given traffic dataset. The properties observed and analyzed are later used to generalize the results to the generation process and the prediction process. In the following sections we will see how Network Networks allow us to produce models that are easily generalized to any traffic and give an accurate prediction of the traffic, while the ARIMA models allow us to flexibly generate traffic so that is has the properties that we desire at each situation.

Chapter 6.

Simulation results

In this section the prediction performance of the polynomial model and the neural model that have been implemented and explained in section 4.2 are tested with the traffic generated through the implementation explained in section 4.1. When the models deal with evolutionary traffic, the Evaluator module, as explained in section 4.3 will give intelligence to the simulator to allow decision on whether there is need to re-estimate the models or not. Different reaction criteria are compared when non valid models are detected, in order to determine the best action strategy, both for better accuracy in the predictions and smaller computational cost of the overall computations carried.

6.1 Modelling non-evolutionary traffic

We now present an example of non-evolutionary traffic generation in the simulator and how the polynomial model would predict the traffic. This traffic is generated with the following equation explained in section 4.1, for a set of 2 nodes and with a simulation time of 40 days:

$$traffic(t) = B(t) + noise(t) \tag{6.1}$$

First the generation of a polynomial model for this traffic is going to be considered. The procedure to produce the polynomial model from an Mdt was explained in section 4.2.1. Recall that we consider each of the three time series saved in the Mdt, and to each of them we calculated the period and proceeded to fit the optimal degree polynomial in the BIC sense. Confidence intervals for the average prediction can be obtained by the standard deviations, or by the minimum and maximum polynomial predictions. Figure 6-1 shows in blue the real traffic observed during one day, and in red, (a) shows the prediction of the maximum traffic, (b) the prediction of the average traffic, and (c) the prediction of the minimum traffic.

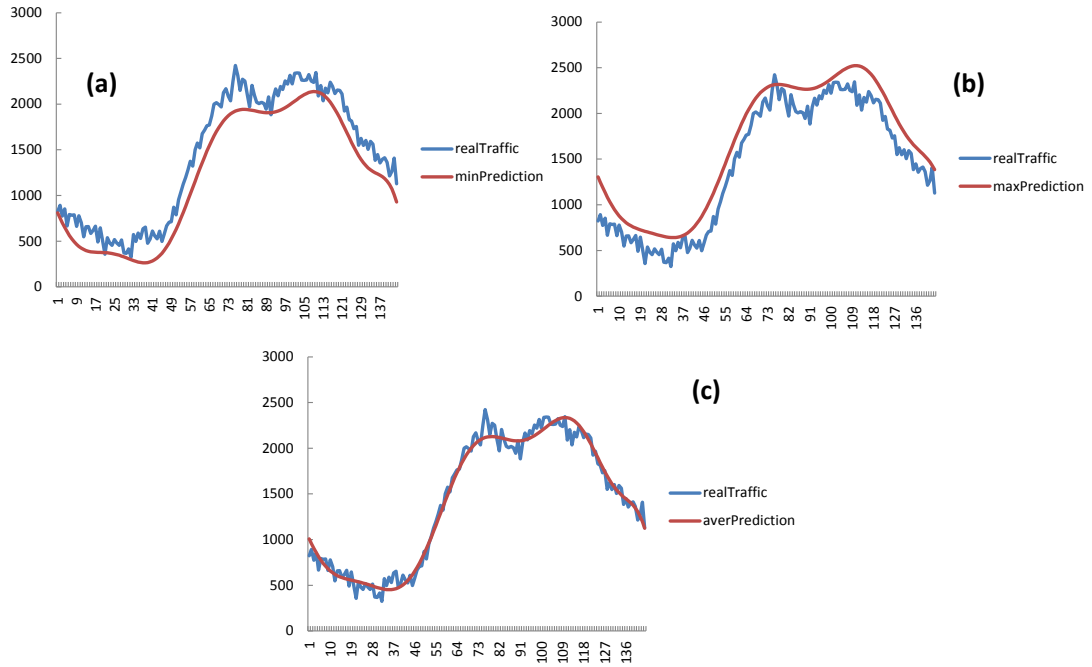

Figure 6-1: Polynomial Model for the simulated traffic

Observing Figure 6-1, we notice that the average traffic is a good prediction of the traffic, while the maximum and minimum polynomials fit well the expected traffic from the upper and the lower part as we expected.

We are now going to show the numerical results obtained by this traffic predictions with the polynomial models. Table 6-1 shows the numerical results of the polynomial average model. In this case we notice that the mean of the residuals of the prediction is almost 0 (as would be expected for a centered prediction) that the maximal error in prediction has absolute values 494.23 and the mean prediction error is of 73.19. Given that the traffic maximum is 2542, we are talking of errors of maximum 20% and mean 3% of the value respectively.

Table 6-2 shows the numerical results of the polynomial minimum model. In this case, we consider only the values that are above the traffic observed (and would mean that the traffic is outside the confidence interval). These points represent 1.82% of the total of predictions made, and their mean is 36.19, which is an error of mean about 1.5%.

Finally Table 6-3 shows the numerical results of the polynomial maximum model. Again only the points that would make the observed be outside the confidence interval are considered. They represent 4.82% of the total of points, and their residual mean is 34.71, which implies an error of more than a 1.5%.

Table 6-1:Numerical results of polynomial average model

Standard deviation	91.90
Residual's mean	0.89
Absolute residual's mean	73.19
Absolute residual's maximum)	494.23

Table 6-2:Numerical results of polynomial minimum model

Standard deviation	6.63
Outer residuals' mean	36.19
Outer traffic percentage	1.82%

Table 6-3:Numerical results of polynomial maximum model

Standard deviation	4.82
Outer residuals' mean	34.71
Outer traffic percentage	1.14%

We are now going to study the performance of the neural model, explained in 4.2.2. It also produces three models, one for the average, one for the maximum and one for the minimum. The idea is also, to predict the future traffic through the average model, having a confidence interval with the maximal and the minimal neural predictions.

We are first going to show an example of the structure determination, explained in Table 4-1. We are taking, for the traffic produced by (6.1), the maximum neural model. We detect a period of 24h with the algorithm *ComputePeriod*. This makes our first structure to be NN(24:1:1). We calculate the testing errors in prediction and keep the sum of squares and the maximum value of these errors. Also, the smallest weight in absolute value is determined, which is lag 8, so in the next iteration, NN(23:1:1) is trained with all initial input lags except for lag 8. This is done until one lag remains, in this case lag 24, and NN(1:1:1) is trained. Figure 6-2 (a) shows the testing errors RSS and (b) shows the testing errors maximal for each NN(nlags:1:1) model being trained.

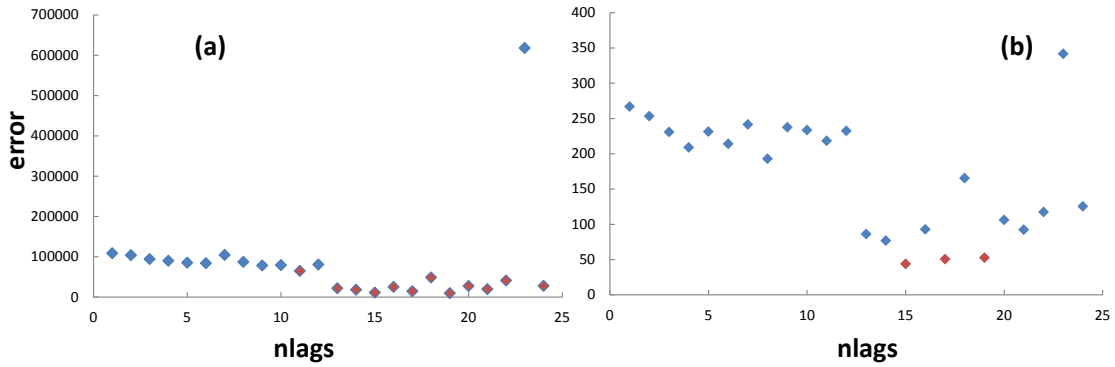

Figure 6-2: (a) Testing errors RSS (b) Testing errors maximum for different number of inputs

Observe that as we decrease the number of input lags, the error increases, reaching its maximum at NN(1:1:1), except for *nlags* = 23, which has a very high error possibly due to the convergence of the training. This means that small number of input lags do not suffice to explain the final traffic. Now we take the minimal RSS testing error, which is achieved at NN(19:1:1) with a value of 9984.31.

We took a = 0.1, and we got a threshold of 70760.5 for the RSS, which meant that the structures that had a good enough RSS testing error were NN(11:1:1), NN(13:1:1), NN(14:1:1), NN(15:1:1), NN(16:1:1), NN(17:1:1), NN(18:1:1), NN(19:1:1), NN(20:1:1), NN(21:1:1), NN(22:1:1) and NN(24:1:1), plotted in red in Figure 6-2(a). Now, from these 12 structures we determine the ones that satisfy that the maximum error in the testing set is also relatively small enough. The same formula used before, but for the maximum maximal error and de minimum maximal error overall, produced a threshold of 73.70. Only the structures NN(15:1:1), NN(17:1:1) and NN(18:1:1) satisfy this threshold. Now that we have the models that are accurate enough for us, by enforcing a threshold for the RSS and a threshold for the maximum error, we simply choose the simplest model. Therefore, in this case, the final number of lags is the smaller from the remaining three, and so 15 lags. The 15 final inputs, obtained by deleting one by one the less important inputs at each iteration are: 2, 5, 7, 10, 11, 12, 13, 14, 16, 17, 19, 21, 22, 23 and 24.

We now proceed to set the optimal number of hidden neurons with the method described above, and threshold minRSS+a(maxRSS-minRSS) with a=0.1, in this case producing a threshold of 29014.7. The structures that satisfy the threshold are NN(15:1:1) and NN(15:13:1), and from this we take the simplest structure, therefore NN(15:1:1). Figure 6-3 shows a plot of the evolution of the RSS in the testing set for different number of hidden neurons in the structures NN(15:nhidden:1).

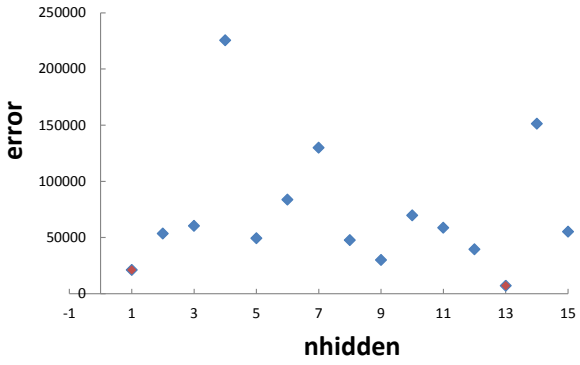

Figure 6-3: (a) Testing errors RSS for different number of hidden neurons

Therefore, after the application of the structure determination algorithm explained in Table 4-1, for the given OD pair and the maximum time series the ANN structure chosen is NN(15:1:1). The same is done for the minimum and average time series, and for the rest of OD pairs.

We are now going to show the numerical results obtained by this traffic predictions with the neural models. Prediction is obtained as a piecewise linear function that has spikes at every entire and half hour, by the procedure explained in 4.2.2. Figure 6-4 displays a one day prediction using this model for the same traffic. Figure 6-4 (a) shows the minimum prediction with the Neural Model, Figure 6-4 (b) the maximum, and Figure 6-4 (c) the average prediction.

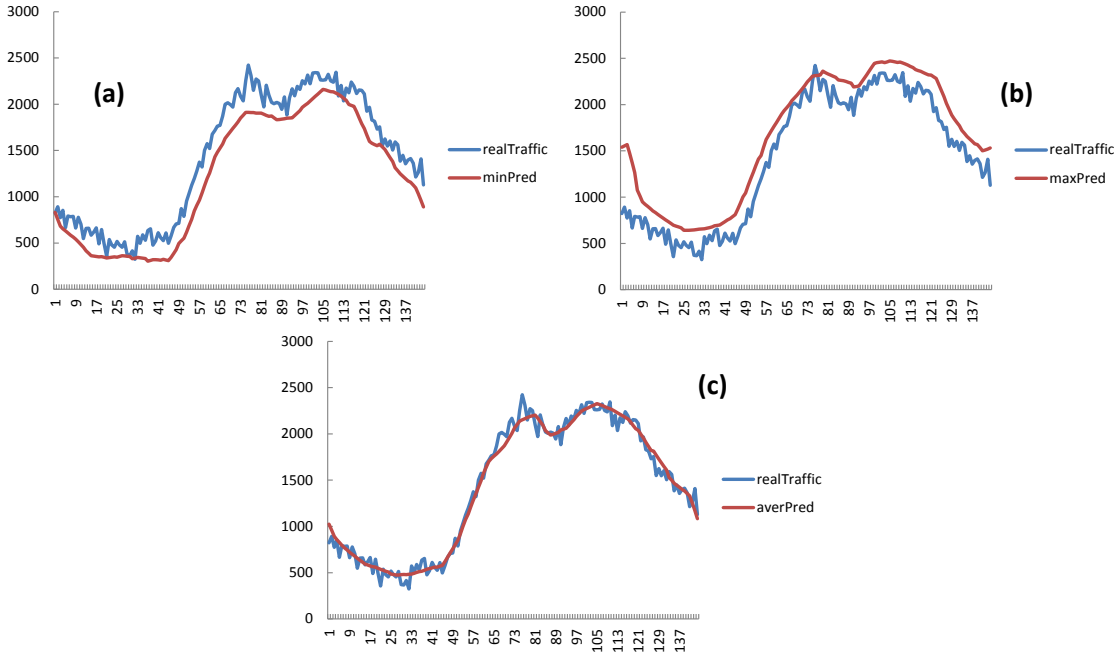

Figure 6-4: Final Neural Model for the simulated traffic

We proceed as in the polynomial model to compute some values in order to compare the goodness of fit between the models. Table 6-4 shows the numerical results of the neural average model. Again we notice that the mean of the residuals of the prediction is almost 0. The maximal error in prediction has now absolute values 498.71 and the mean prediction error is of 67.68, which is smaller than the polynomial model, meaning that the prediction is more accurate. Table 6-5 shows the numerical results of the neural minimum model. Again, we consider only the values that are above the traffic observed, which represent 3.71% of the total of predictions made, and their mean is 60.22. Finally Table 6-6 shows the numerical results of the neural maximum model. Again only the points that would make the observed be outside the confidence interval are considered, which represent 0.80% of the total of points, and their residual mean is 30.62.

Comparing both models, one can notice that none is significantly better than the other when the traffic keeps the same profile and intensity through time, even if the average neural model produces fairly more accurate predictions.

Table 6-4:Numerical results of neural average model

Standard deviation	85.17
Residual's mean	-1.33
Absolute residual's mean	67.68
Absolute residual's maximum	498.72

Table 6-5:Numerical results of neural minimum model

Standard deviation	15.98
Outer residuals' mean	60.22
Outer traffic percentage	3.71%

Table 6-6:Numerical results of neural maximum model

Standard deviation	3.82
Outer residuals' mean	30.62
Outer traffic percentage	0.80%

6.2 Modelling evolutionary traffic

We proceed to produce predictive models for evolutionary traffic. First of all we consider a traffic with increasing intensity, that is, starting at the initial daily profile, *B(t),* and linearly evolving during 40 days to 5*B(t)*, which is generated by the following formula explained in section 4.1:

$$traffic(t) = \alpha(B(t) + noise(t)) + (1-\alpha)(5B(t) + noise(t)) \quad (6.2)$$

Models are generated during the warmup period, which is set to be the first 5 weeks of the simulation and afterwards traffic is predicted either using the polynomial model or the neural model. Because of the evolution of traffic throughout the simulation, it will be expected that the errors in prediction increase daily.

If tendency and amplitude is not considered, the polynomial prediction, by definition is non evolutionary, and it remains the same regardless of the traffic observed after the warmup period. Figure 6-5 shows the 35 day average prediction, in red, using the obtained polynomial model with the 5 first days of the simulation (warmup period), versus the real traffic in blue.

If we now apply the algorithms in Table 3-2 and Table 3-3 to preprocess the amplitude change and the tendency, the amplitude algorithm is able to cope well with the evolution of the traffic, since the traffic was generated exactly like this. This produces a polynomial model that is able to predict much better the changes in traffic and evolves adequately with time. Figure 6-6 shows the 35 day average prediction, in red, using the obtained polynomial model, applied to using the preprocessing in amplitude and tendency, with the 5 first days of the simulation (warmup period), versus the real traffic in blue.

Finally we study how the neural model performs with this traffic. Due to the model itself, the many parameters and hidden layer, it is not intuitive how it acts when traffic changes. Figure 6-7 shows the 35 day average prediction using the obtained neural model, in red, trained with the Mdt of the 5 first days of the simulation (warmup period), versus the real traffic in blue. Notice that it is able to adapt quite well to changes, probably because traffic is predicted by last observed traffic, and the relationships between these observations remain similar even when the traffic evolves.

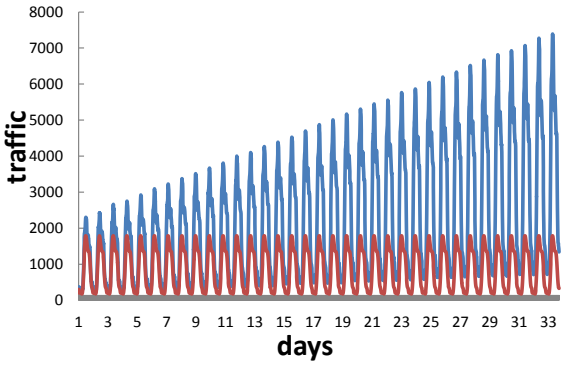

Figure 6-5: Traffic with intensity change with no preprocessing versus polynomial average prediction

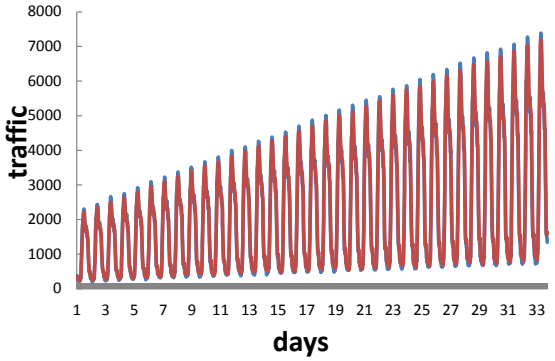

Figure 6-6: Traffic with intensity change with amplitude and tendency preprocessing versus polynomial average prediction

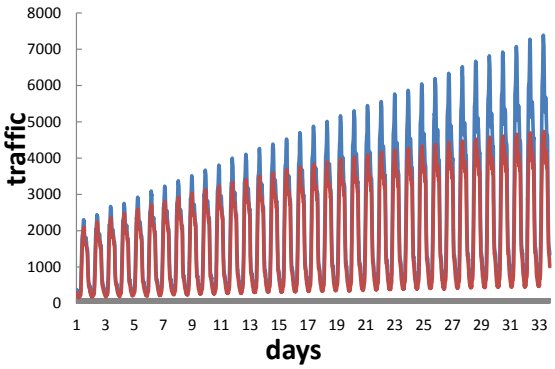

Figure 6-7: Traffic with intensity change versus neural average prediction

To show the evolution of the daily RSS we have represented them in Figure 6-8. (a) shows the neural model RSS in blue against the polynomial model with no preprocessing in red, and (b) shows the neural model RSS in blue against the polynomial model with preprocessing in green, and the y value is divided by 10000. Figure 6-9 plots the same but for the maximal error in the prediction, (a) contains the daily maximum error in blue of the neural model and in red of the polynomial model with no preprocessing and (b) the daily maximum error in blue of the neural model and in green of the polynomial model with preprocessing.

It can be seen that the neural model is the one making the most accurate predictions up to day 15 of the simulation with smallest RSS, which is day 10 in the plots (recall the first 5 days are the warmup period), and so when the traffic is up to 200% of the training traffic, but fails to give good predictions at the end of the simulation, when traffic is 500% of the training traffic. This means that the neural model is able to adapt to the traffic change until a certain point, while polynomial model without preprocessing, by its definition, remains non evolutionary. On the other side, from day 15 and on, the polynomial model with preprocessing is able to maintain reasonable errors, with a much slower increase than the other two models.

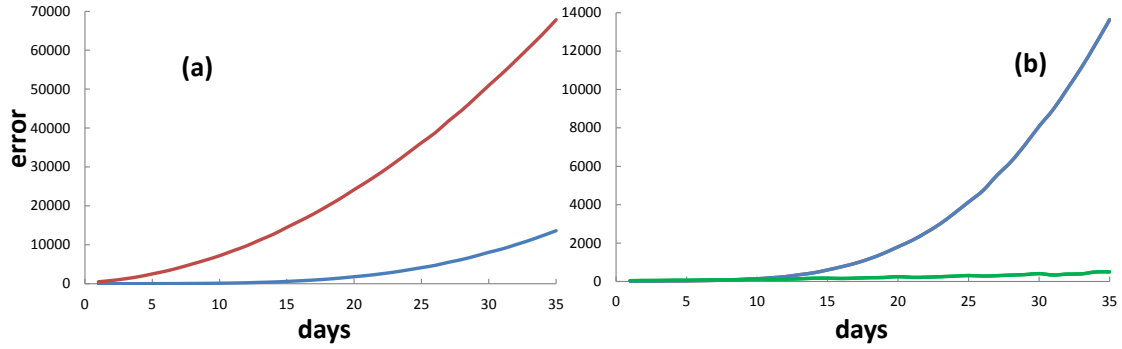

Figure 6-8: RSS of (a) neural model in blue vs. polynomial model with no preprocessing in red and (b) vs. polynomial with preprocessing in green

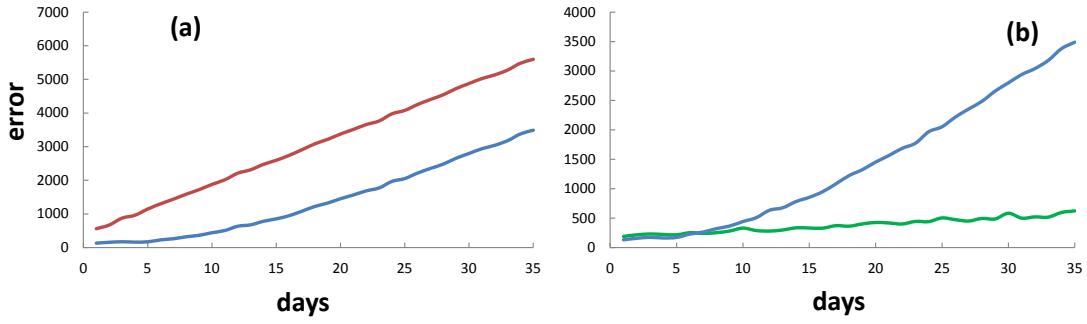

Figure 6-9: Maximum error of (a) neural model in blue vs. polynomial model with no preprocessing in red and (b) vs. polynomial with preprocessing in green

The next step is to consider a traffic that evolves during 20 days to a different daily profile, generated by the following formula, explained in 4.1:

$$traffic(t) = \alpha(B(t) + noise(t)) + (1-\alpha)(CDN(t) + noise(t)) \quad (6.3)$$

Predicting traffic with the polynomial model will obviously lead to bad results: the polynomial is fitted with the first traffic profile, and therefor will predict the last days of the traffic exactly as the initial traffic profile *B(t)*, and not as the second profile *CDN(t)*. Figure 6-10 shows the plot of this real traffic with changing profile during 20 days (recall that the first 5 days are not plotted, since they are the warmup period used to create the model), in blue, versus the average prediction performed by the polynomial model.

We now proceed to do the same with the neural model, and see if passed observations from the second profile are able to produce accurate predictions even if the model was trained with the first profile. Figure 6-10 shows the plot of the same real traffic with changing profile during 20 days, in blue, versus the average prediction performed by the neural model.

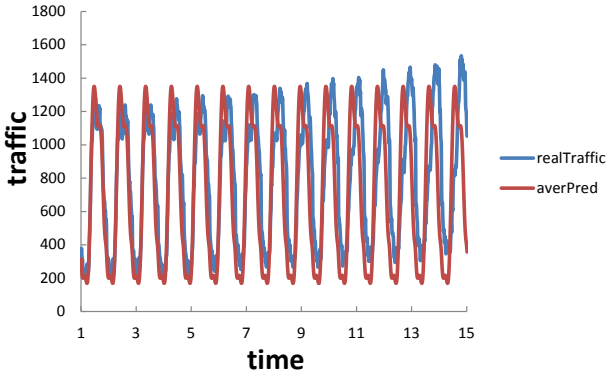

Figure 6-10: Traffic with profile change versus polynomial average prediction

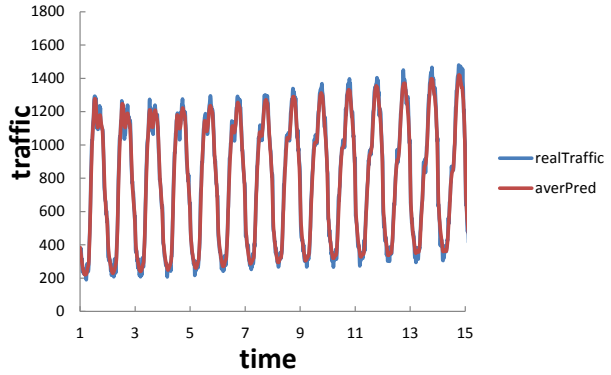

Figure 6-11: Traffic with profile change versus neural average prediction

There are no significant differences if we preprocess data for the polynomial model or not, since there is no trend and traffic is already homoestatic since the noise added comes from the same distribution.

We now proceed to compute to daily predictive errors of this traffic predictions. Figure 6-12shows the daily RSS (a) and maximum error (b) of the polynomial model, in blue, against the neural model. The polynomial model has an important increment on the error as time passes, while the neural model has always smaller errors, and increments with time as expected, but with no significant differences.

It can be observed that the neural model adapts pretty well to the profile change. To have a better understanding of the situation. Figure 6-13 (a) plots the prediction of the neural model in blue during the first days (day 6 in the simulation) and (b) the prediction of the neural model in red in the last day of the simulation versus the observed values of traffic in red. It can be observed that the first prediction is much more accurate (since the traffic profile is very similar to the training traffic profile), while the last prediction is less accurate since now the traffic has a different profile from the training traffic. Nevertheless, the prediction is still quite accurate, meaning that the neural model adapts well to profile changes.

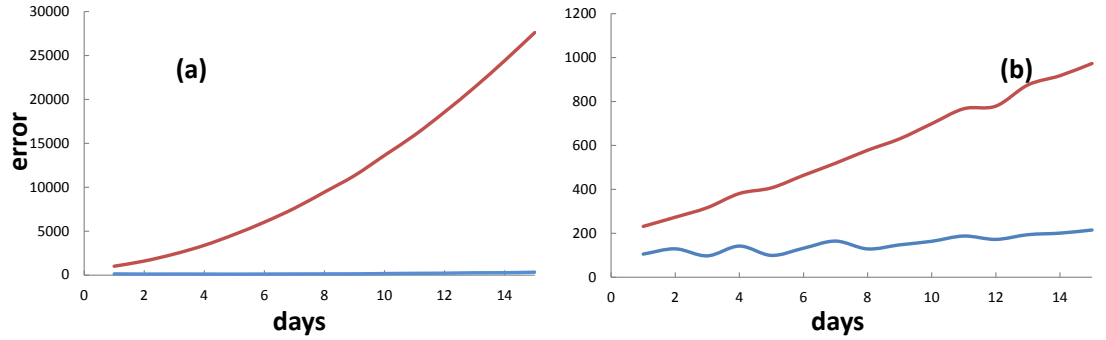

Figure 6-12: RSS errors (a) and maximum error (b), of polynomial model in blue versus neural model in red

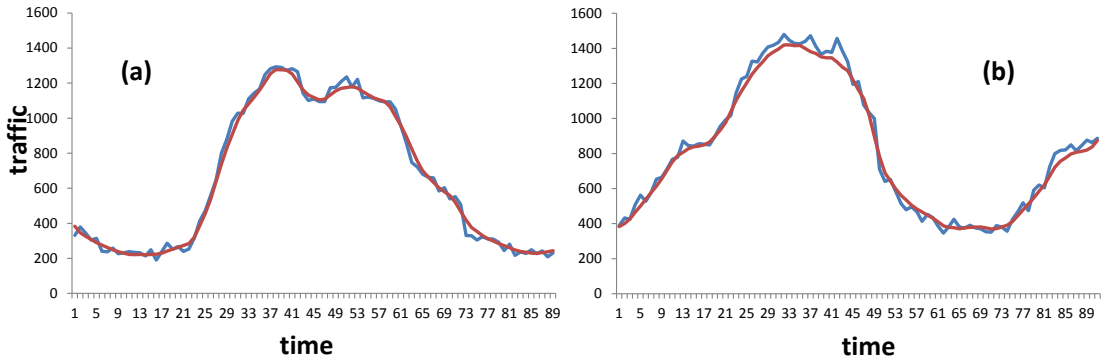

Figure 6-13: Neural model prediction in blue versus observed traffic (a) in day 6 (b) in day 20

Next step is to combine traffic changes in intensity with traffic changes in profile, starting with the profile *B(t)* and linearly evolving to *5CDN(t)* in a 2 node network and during a 40 days simulation, generated by the following formula, explained in section 4.1:

$$traffic(t) = \alpha(B(t) + noise(t)) + (1-\alpha)(5CDN(t) + noise(t)) \quad (6.4)$$

We again proceed first to obtain the polynomial model, applied without preprocessing of data, which is shown in Figure 6-14 (average prediction plotted in red, observed traffic plotted in blue). Next, we consider the polynomial model with preprocessing of the tendency and amplitude, which will be able to cope these changes, but not the change on the profile. This average prediction is shown in Figure 6-15 (average prediction plotted in red, observed traffic plotted in blue). Finally we consider the neural model generated from this traffic, whose average prediction is shown in Figure 6-16 (average prediction plotted in red, observed traffic plotted in blue).

As expected, the polynomial model without preprocessing remains non evolutionary throughout time, while the traffic changes, making the predictive errors increase rapidly with time. As for the polynomial model with preprocessing, we notice that it produces adequately the change in amplitude and tendency, but despite this improvement, the profile will not be able to be adapted, since the profile remains by definition of the model, as was generated during the warmup period, and so, with the first profile *B(t).* This model will be, therefor, very similar to the model plotted in Figure 6-6, since the generated traffics during the warmup period are very similar, as in this case, the profile is basically generated by *B(t).* Nevertheless, errors in prediction will increase in this case in respect to the other model because of the profile evolution.

Finally, studying the neural model, similarly to the model created for traffic that changed profiles, this model adapts well to changes in profile, and similarly to the model created for traffic that changes intensity, the model adapts well to intensity changes up to a certain point (about up to 20a0% the original traffic). Recall again that this is an extreme example where the final traffic is 500% the starting traffic, which means that in normal situations, this model will be able to predict pretty well the future behavior of the traffic.

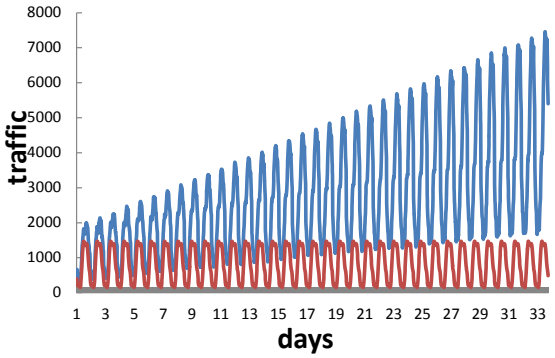

Figure 6-14: Traffic with profile and intensity change, in blue, versus polynomial with no preprocessing average prediction, in red

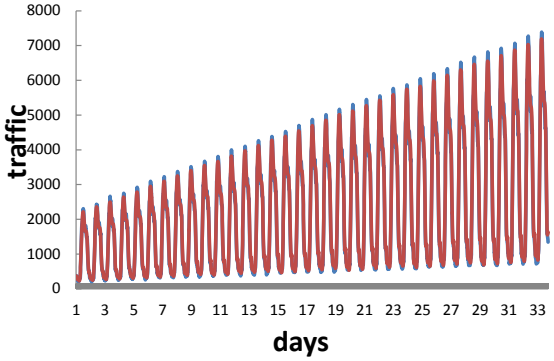

Figure 6-15: Traffic with profile and intensity change, in blue, versus polynomial with tendency and amplitude preprocessing average prediction, in red

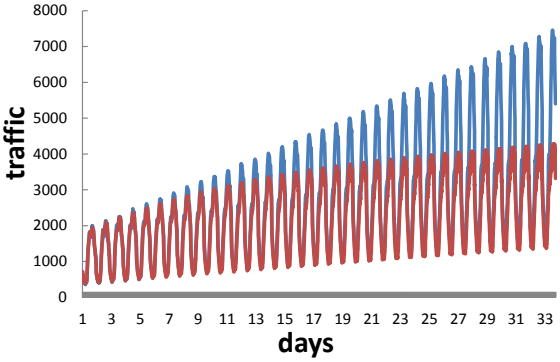

Figure 6-16: Traffic with profile and intensity change, in blue, versus neural average prediction, in red

To better understand how the neural model evolves in time, Figure 6-17 shows two consecutive days of the prediction. (a) shows days 6 and 7 of the simulation, the first days after the warmup period, and almost perfect fitting can be observed, due to the fact that traffic is very similar to the training traffic of the model. (b) shows days 10 and 11 of the simulation, where traffic is produced following $traffic(t) = 3/4(f(t) + noise) + 1/4(5g(t) + noise)$, and so still a quite good prediction is performed. Finally, (c) shows the last two days of the simulation, where traffic is $traffic(t) = 5g(t) + noise$, and since it is much higher than the training traffic, the prediction in the highest parts is far from the observed values.

Finally we calculate the daily errors of both models. Figure 6-18 shows the neural model RSS in blue against the polynomial model with no preprocessing in red, and (b) shows the neural model RSS in blue against the polynomial model with preprocessing in green, and the y value is divided by 10000. Figure 6-19 plots the same but for the maximal error in the prediction, (a) contains the daily maximum error in blue of the neural model and in red of the polynomial model with no preprocessing and (b) the daily maximum error in blue of the neural model and in green of the polynomial model with preprocessing.

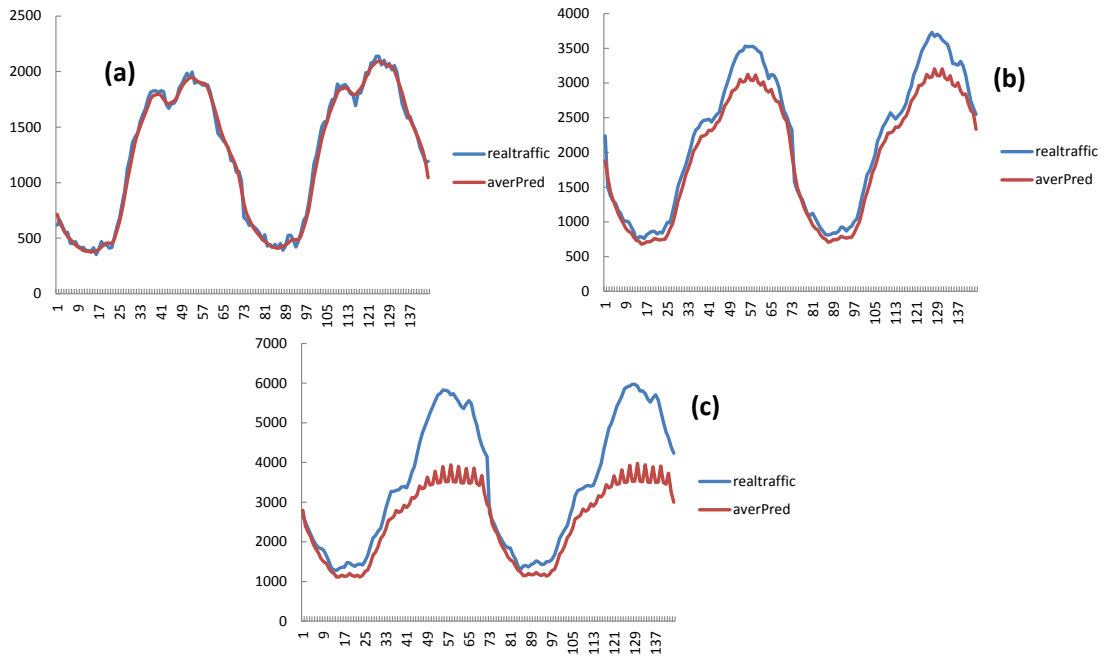

Figure 6-17: Traffic with profile and intensity change versus neural average prediction in days (a) 6 and 7 (b) 10 and 11 and (c) 39 and 40 of the simulation

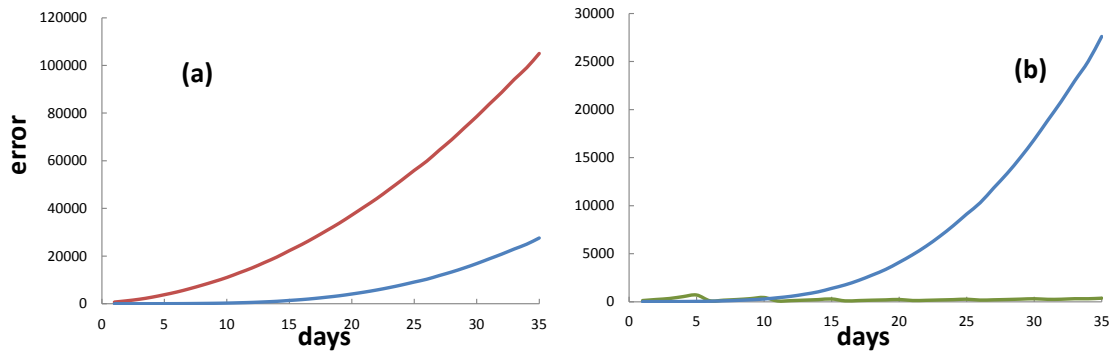

Figure 6-18: RSS of neural model in blue vs. (a) polynomial model with no preprocessing in red (b) polynomial with preprocessing in green

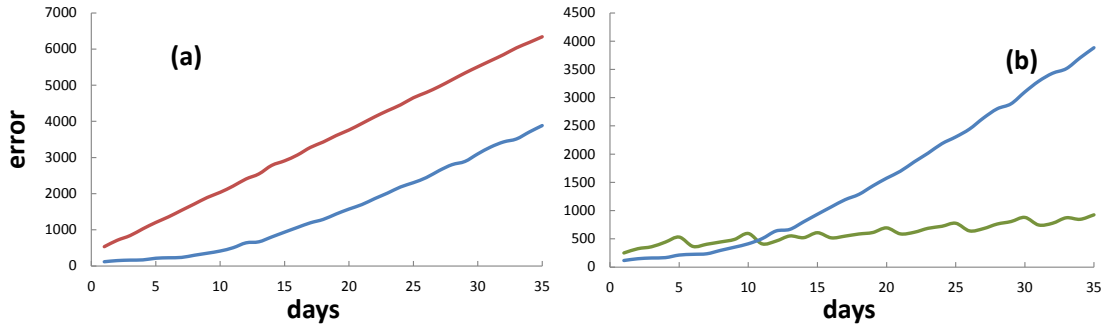

Figure 6-19: Maximum errors daily errors of (a) neural model in blue vs. polynomial model with no preprocessing in red (b) neural model in blue vs. polynomial with preprocessing in green

The plots are similar to the ones obtained for the traffic that only changed in intensity, because the biggest change in this traffic is due to the intensity change. Nevertheless, errors here are much bigger than in the other case, because here we are considering more change in the traffic. Up to day 16 of the simulation (11 in the plot), the neural performs more accurately than the polynomial model, and from that day the model that performs more accurately is the polynomial model with preprocessing.

6.3 Recomputation of models

We have seen that the neural model performs good prediction when the traffic changes up to a certain degree of change. On the contrary, the polynomial model remains non evolutionary, as would have been expected by its definition, or it adapts well if the preprocessing is lucky enough. Still, in order to have valid models at any given time in the simulation, recomputation of models needs to be performed, since if the traffic has some sort of evolution, the models will stop being

valid at some point in time. One could set this recomputation to be done periodically every given time slot.

In order to better understand how this would work, we have implemented a recomputation of the models every 10 days of the simulation for the neural model. Figure 6-20 shows the prediction of these set of neural models that are computed periodically every 10 days, and we can compare it to Figure 6-16, which showed the prediction of the neural model for the same traffic without recomputation.

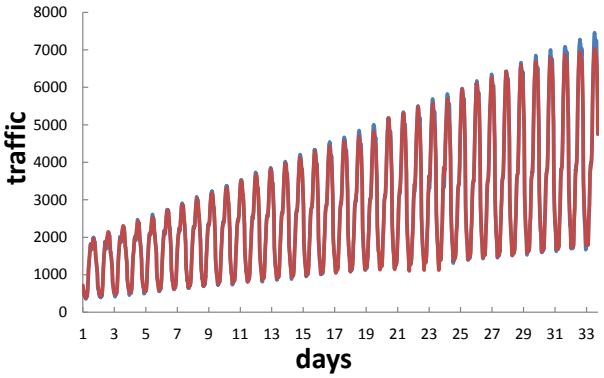

Figure 6-20: Traffic with intensity and profile change versus periodically computed neural average models prediction

Recomputing the models periodically leads to high computational costs that can be avoided in case the models are still valid at the next time slot. For this reason, we have created the module Evaluator, that determines the goodness of fit of the models, and return to the Analytics module which of the models are still valid and which are no longer valid, and from this the Analytics module is able to decide if the model for that OD to be recomputed or not.

Recall that we saved our passed traffic into the Mdts, which contains the hourly minimum, maximum and average. The models we produced, PolynomialModel and NeuralModel, had some methods applied in order to obtain a smooth prediction for any given time, regardless to the fact that the models where produced form discrete hourly values. Nevertheless, the goodness of fit computation is related only to the values of the Mdt, and so the ones that are producing the models.

In order to determine if a model is valid or not, we use the fitness evaluation algorithm explained in Table 4-2. First the current Mdt for the given OD pair is considered. We use the average prediction of the traffic to compare it with the observed traffic. Both Neural Model and Polynomial Model produce the expected values that are used to compute the statistic λ. From this statistic and the degrees of freedom, we can compute the p-value, and we can say with a degree of certainty that the model fits well the data if the p-value is smaller than a threshold α. Otherwise, we are in the case where the model is not good anymore, and so the model for this OD pair should be computed again.

For instance, in the case of a neural model, where we have a degree of freedom of 95, since we have 5 days saved in the Mdt producing 120 hourly values and the first 24 cannot be predicted if the ANN has a maximum input lag of 24. The model is considered to be valid while the statistic value is smaller than 125, with a p-values bigger than 0.05.

6.3.1 Evolution on the number of non valid models under realistic traffic

Up to this section extreme situations of traffic evolution have been tested to better understand how the models would adapt to different traffic evolutions.

We proceed to study a more realistic scenario, with a bigger number of network nodes, 10, giving a total of 90 OD pairs. Also we take a more realistic evolutionary traffic, and bigger noise producing a bigger degree of randomness on the final traffic. The traffic we are considering is the following, where the initial profile is doubled throughout 700 days:

$$traffic(t) = \alpha(B(t) + noise) + (1-\alpha)(2B(t) + noise) \tag{6.5}$$

Where *noise* follows N(0.5,3) and is scaled at 10.

We proceed now to study how the number of valid models and their statistics evolve with time as traffic changes. Since there is evolution of traffic, we are only going to consider neural models for this traffic, as polynomial models will give too high statistics even in the first days of prediction, as they are not, in the beginning as accurate as the neural model.

Decision on the structure of each of the 270 ANN, 3 for each of the 90 OD pairs, is done by the algorithm explained in Table 4-1, and the final structure of each of these is trained and saved as the final model. Given that we are training a very big number of NN, between 6480 and 12960 (24 for the number of input lags, from 1 to 24 for the number of hidden lags, for each of the 270 NN), this takes a lot of computational time: 6.37 hours.

At this point, every 10 days, the Evaluator module calculates the statistic for each OD pair average model. Figure 6-21 shows the evolution of the number of non valid models as time advances, that is the number of models with a statistic that produces a p-value smaller than 0.05. As expected, as time advances and traffic gets more different from the traffic during the warmup period, the number of non valid models increases. But still until the day 345 more than 50% of the models are still valid, and at the end of the simulation, 30% of the models are still valid. What this means is that, even if the computation of the models is quite expensive, these models are very accurate making predictions for a long time with a reasonable traffic evolution. Figure 6-22 shows the evolution of the statistic's mean at every evaluation.

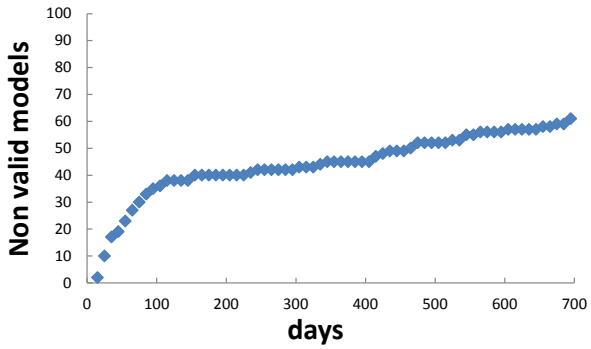

Figure 6-21: Evolution of the number of non valid models at every evaluation

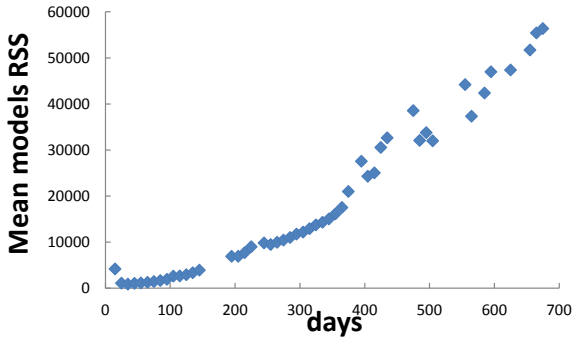

Figure 6-22: Evolution of the statistics' mean at every evaluation

Notice that the increment on the statistics' mean is much steeper than the increment on the number of non valid models. That is because the non valid models produce very bad predictions as time advances and this leads to a bigger increment on the statistics. Also there are some statistics means missing, because there was a division by zero when computing one or more of the non valid statistics due to an average prediction of value 0.

Even if the performance of the produced models is good for a quite long time, there is an obvious problem: the great expense of training all the structures for the 270 final models that want to be obtained. In order to obtain the models faster, we are next going to find the optimal considering only NN(24:1:1) and NN(24:2:1) for all the OD pairs

Since this decision compares less models, the accuracy of the predictions will be worse, leading to bigger statistic means and to bigger number of non valid models. The same traffic has been simulated, and Figure 6-23 shows the increase on the number of non valid models as time advances. On the 115th day of the simulation already all the 90 models are not valid, and this is maintained until the end of the simulation. Also on the first evaluation 15 models are already non valid. Generating the models takes now, though, only 19.7 minutes.

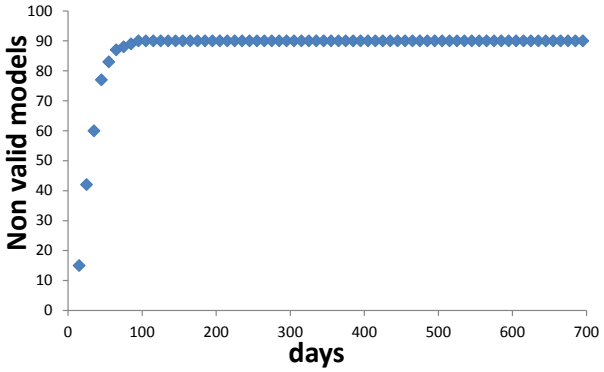

Figure 6-23: Evolution of the number of non valid models at every evaluation

6.3.2 Periodic recomputation of all the models

To avoid the increase of inaccuracy in the predictions when time advances, an obvious solution would be to recompute periodically, say every given recomputation period that we take to be 10 days, all the models. Doing this testing all possible structures for the models would be far too computationally expensive (computation of the models would take more than 17 days). Still, if the recomputation period is small enough, models do not need to endure so well changes, and we do not need to have such optimal models. We are therefore going to test this idea again with the same fixed structures NN(24:1:1) or NN(24:2:1). This had a total cost of 22.98 hours.

Recomputing the models every 10 days we are ensuring that there is always a small number of non valid models, and this implies that the predictive performance of the majority of the models is accurate. Figure 6-24 shows how now the number of non valid models does not increase as traffic changes, since the models are now computed with the latest traffic saved, The number of non valid models is always within 1 to 13, i.e. always more than 85% of the models perform accurately, and a mean of 6 non valid models.

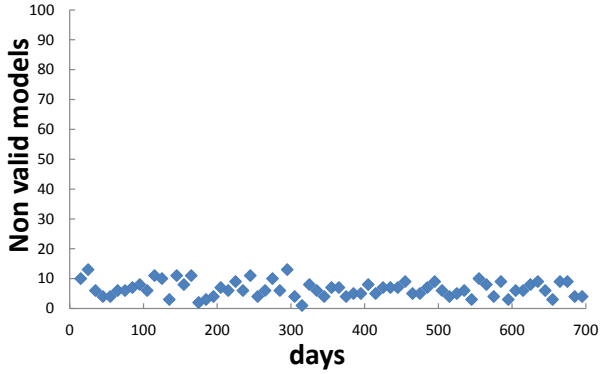

Figure 6-24: Evolution of the number of non valid models at every evaluation

Keeping the number of valid models as high as possible throughout the whole simulation is very important in order to ensure that predictions are performed accurately. But the recomputation of all the models every recomputation period makes the overall process very expensive.

6.3.3 Threshold based periodic recomputation of all the models

Improvements on the decision of recomputing the models can be done following the fact that traffic in each OD pair is not independent from the rest. This implies that, even if some OD pair has still a valid model but the others do not, there is an obvious change of traffic, and probably next time the model will not be valid anymore. Therefore, the decision can be taken in the overall network instead of for each given OD pair. The idea is that, if the percentage of pairs with an invalid model is higher than a threshold, then we perform recomputations of all the models, because this tells us traffic has changed significantly, and the rest of the models that are still valid will soon not be valid either. The same can be applied to a given node: if the percentage of models of the traffic that leave the node is bigger than a threshold, all the models of that node are recomputed.

We finally propose a threshold based strategy, which recomputes all the models at every evaluation but only if this evaluation tells that the number of non valid models is bigger than a given threshold. Figure 6-21 was an example of this threshold based strategy with a threshold of 1 invalid model, with a mean of 6.5.

For instance, if we consider a model to be valid if the chi-squared p-value is smaller than 0.05 and we recompute all the models when there are more than 7 invalid models, the resulting simulation computes new models 57 out of the 69 times the evaluator module is called. This makes the number of non valid models to be within 4 and 34, with a mean of 12.52. The evolution of the number of non valid models can be observed in Figure 6-25, in green plotted the evaluations were no recomputation was performed as the threshold was not passed.

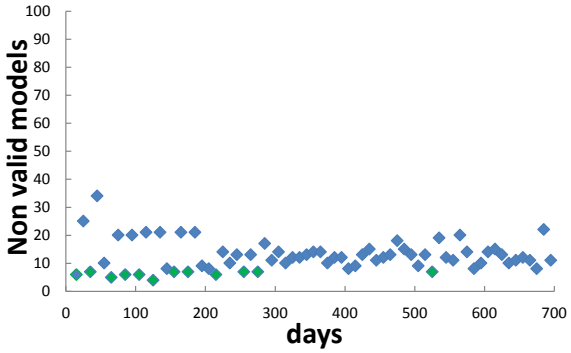

Figure 6-25: Threshold based evolution of the statistics' mean at every evaluation

By generating the models an 82.6% of the times, we have decreased almost 20% of the computational costs, while maintaining a small number of models that do not pass the test. This can still be improved noticing that many of the models that are being recomputed at each recomputation period were still valid, and therefore did not need to be recomputed.

6.3.4 Periodic recomputation only of invalid models

What is being proposed here is to periodically evaluate the fitness of the models, and recompute them only when the fitness is no longer good enough. This results to less expensive computational costs in case the model doesn't need to be recomputed at every checking time, since evaluating the model is much less expensive that recomputing the models. The overall computational time of training all the models until the end of the simulation is now of around 2 hours. That means, we have reduced to the 10% of the original cost.

This is a reactive strategy that saves a lot of computational cost, but has also an important downside, results are not as good as when generating all the models at every evaluation. Figure 6-26 shows the evolution of the number of non valid models at every evaluation, when the recomputation of models is done only to the models that were not valid in the evaluation. This number of non valid models is kept between 14 and 33, with a mean of 24.

We can also compare the statistic's means at every evaluation in this performance, to the means in the other performance. We again observe, as would have been expected values with no significative trend (since recompuatation of models is also done periodically), but the values are much bigger than before, with an overall mean of 116.15.

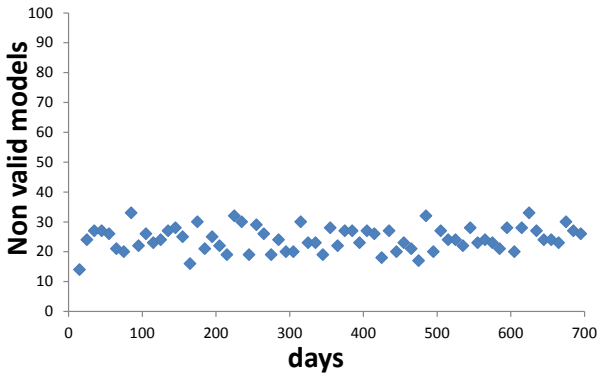

Figure 6-26: Evolution of the number of non valid models at every evaluation

This increase on the number of non valid models implicitly indicates an increase on the predictive errors of the models and this tells us that it is not a good strategy to recompute only the models that failed the hypothesis test, because it is very likely that even if they passed the test, at the next evaluation will fail to do so, and for some days these models will be producing inaccurate predictions.

6.4 Conclusions

Throughout this section we have seen how the polynomial model and the neural model are computed from a given Mdt, and how each of them perform to some traffic with different properties.

We first considered non evolutionary traffic, and noticed that both models performed similarly, but prediction by the average model was more accurate in the case of the neural model.

When considering evolutionary traffic we differentiated between intensity change, profile change and both intensity and profile change. The polynomial model without preprocessing, as was expected by its definition, remained non evolutionary throughout time for the three cases, and therefore its errors in prediction increased daily. When tendency and homoestecity was considered we observed an important improvement when only intensity change was considered, on the predictions of the polynomial model, leading actually to the best performance of the three models considered. It had, nevertheless, a drawback: this improvement on performance was due to the very particular way the evolution was taking place, but the model would not be able to predict as accurately to general types of evolutionary traffic, as for the traffic with profile change.

The neural model acted differently. We observed that when dealing with profile changes it adapted well and produced good predictions, and when dealing with intensity changes and both intensity and profile changes it adapted well until a certain point (when increment in the initial traffic was less than 200%), and afterwards failed to make good predictions. In the three cases, it was the model that at least until day 15 of the simulation, was producing the prediction the most accurately.

This means that, while the neural model can be used to predict well traffic when there are reasonable intensity changes (such as sporadic days where traffic increases due to an event), the polynomial model without preprocessing does not adapt to changes. It can be used, nevertheless, to determine if there is a change on intensity or profile, and be able to decide whether recomputation of models needs to be performed or not.

As for the polynomial model with preprocessing methods, it is able to perform better predictions for some determined evolutionary traffics, but it will still fail

when evolution of trend or amplitude are not constant, or when the periodic profile changes. So even if in this case the prediction is very accurate, this is mainly because the evolution was an intensity multiplicative linear change that the algorithm compute amplitude was able to cope with. But for many other types of evolution of traffic model will not be able to adapt well and fail to produce a good prediction.

The next table summarized the predictive performance of the considered models:

Table 6-7: Accuracy of predictive models for different traffic evolution scenarios

	No evolution	*Intensity increment*	*Profile Variation*	*Both*
Polynomial Model	Yes	No	No	No
Polynomial Model (preprocessing)	Yes	Yes	No	No
Neural Model	Yes	Yes (moderated increment)	Yes	Yes (moderated increment)

The Neural model, other than its proved accuracy also has the advantage that does not require preprocessing and is able to adapt to traffic changes and evolutions. These made us choose this model as the general predictive model in the simulator.

Different computational criteria were performed and checked to decide which would lead to best accuracy in the neural models while keeping computational costs low. The next table summarizes this study, considering different percentages of the number of models that do not pass the hypothesis test:

Table 6-8: Performance of each recomputation criteria for the neural models

	Average % of non valid models	*Computational time (hours)*	*Performance w.r.t. no recomputation*	
			non valid models reduction (%)	*Computational time increase*
No recomputation	97.11%	0.33	0%	x 1
Periodic recomputation	6.67%	22.98	93.1%	x 69.63
Threshold based recomputation	13.91%	19.04	85.7%	x 57.69
Non valid ODs recomputation	26.67%	2	72.5%	x 6.06

When considering realistic evolutionary traffic we observed that optimally deciding the best structure for the neural model lead to quite good results, with more than 30% of the models still valid at the end of the simulation. Nevertheless, the computational costs for doing this were very big, and we proposed to only train the structures NN(24:1:1) and NN(24:2:1), giving us less accuracy but much less computational cost (just 0.33 hours) that would allow us to recompute the models at some given times of the simulation.

Studying the results of recomputation of models, the best performance was produced when recomputing all the models periodically, maintaining the number of invalid models between 1.11% and 14.44%. Decreasing the computational costs of this approach was first dealt with threshold based methodologies, which periodically recomputed all the models only if the number of invalid models was bigger than a threshold. The reduction on the cost was not significant, but this allowed us to be able to put a maximum on the number of invalid models, which can be useful in the simulator.

Recomputing only the invalid models periodically leads to a much better performance of the models while only incrementing the computational time to a 6.06 ratio of the initial time without recomputations. The costs are even smaller than the original 6.37 hours of optimally computing the neural models with the NN Structure Algorithm (with a reduction of 68.60% of the costs). At the same time, this reduced the first mean of invalid models at almost one half, proving that intelligently deciding which models to recompute leads to good performance and small computational costs.

Chapter 7.

Concluding Remarks

7.1 Contributions and work impact

The objective of this project has been achieved by means of two main contributions. Firstly, traffic generation models have been proposed as important tools to analyze and characterize continuous, dynamic, and heterogeneous traffic typical of new telecom services and applications. The statistical analysis of a real traffic data set served as a base to propose different generation processes and models.

The design based on the combination of simpler functions facilitates its integration in network simulators. In this regard, by implementing the models in an OMNet++-based simulator, we have proved its usefulness to flexibly generate traffic following a wide range of characteristics.

This takes us to the second contribution, the study of prediction of future traffic which can be later on applied to network reconfiguration. To that end, different predictive models have been proposed. Evaluation tests have also been carried out to study their validity under certain types of evolutionary traffic. All these methodologies have been implemented into the simulator and have proved to lead to accurate predictions. Several reaction criteria have been also proposed and implemented in order to add some intelligence to the simulator to decide when it is necessary to re-estimate the models. By this means, it is possible to keep the majority of models accurate even if traffic has evolution with reasonable computational costs.

This project has been developed as part of the research of the Optical Communications Group (GCO) of the UPC, within the framework of a GCO-AC scholarship I was granted for the academic year 2015-2016. With the work carried throughout the project, some capabilities have been added to the network simulator by adding the implementation of the algorithms discussed, the estimation and the evaluation modules and some parts of the generation process.

Part of the work carried throughout this project and the applications derived from the models, methods, and implementations developed, have been disseminated in various research publications, thus proving the utility and projection of the work done. Here follows a relation of these papers, in which I am an author of the first two, while in the third one the authors used some of the models in this project for their work:

- A. P. Vela, **A. Via**, F. Morales, M. Ruiz, and L. Velasco, "*Traffic generation for telecom cloud -based simulation*" accepted in IEEE International Conference on Transparent Optical Networks (ICTON), 2016. This paper includes part of the processes, models and implementations for the generation of network traffic done in the project.
- A. P. Vela, **A. Via**, M. Ruiz, and L. Velasco, "*Bringing Data Analytics to the Network Nodes*", accepted in European Conference on Optical Communication (ECOC), 2016. This paper applies some of the processes and models proposed and implemented in this project in order to detect anomalies in the network traffic.
- F. Morales, M. Ruiz, and L. Velasco, "*Virtual Network Topology Reconfiguration based on Big Data Analytics for Traffic Prediction*", in Proc. IEEE/OSA Optical Fiber Communication Conference (OFC), 2016. This paper uses some of the predictive models in order to reconfigure the virtual network topology.

The currently implemented simulator allows, therefore, many possibilities of work. The generation and prediction capabilities of the simulator have many applications, as shown in the papers, that go from detecting anomaly traffic to intelligently deciding how to reconfigure the network to guarantee a good performance of the network.

7.2 Personal Evaluation

Throughout this project I have been able to apply many of the fields I had studied during the Bachelor and the Master. I had also deep interest in combining mathematics, statistics and coding in order to obtain useful results. Moreover, when starting the Master, I was interested in being introduced to the fields of Data Analysis, Artificial Intelligence, Simulation and Big Data, thus taking courses such as *Data Analysis and Knowledge Discovery, Multivariate Data Analysis* and some courses of the *III Big Data School*. For all this reasons this project couldn't have been more rewarding and has made me ensure I want to continue learning and working on related fields.

This project also enabled me to participate in research activities the GCO was conducting. The profits I benefitted from being taught and advised by this group of

researchers are hard to summarize. Starting from my introduction to the reading and contrasting of research papers, the study of some Statistics topics that I had not covered during regular university courses, the introduction to the exciting field of Machine Learning and finishing in the improvements on programming (mostly in R and C++) and modeling skills, as well as in the capacity to solve problems by myself or together with the advisors or some other members of the GCO.

But other than the more academically oriented profits, their constant implication on the project, the newly proposed ideas and the availability and willingness to discuss and solve doubts made me enrich me personally. Being able to participate in the internal workshops and attend to some of the PhD thesis lecture of group members has made me see the importance and quality of the research that is carried, as well as the hard work behind a research group. All this has risen my interests and knowledge about many fields that I would not otherwise have known.

Acronyms

ABNO	Application-based Network Operations
ACF	Autocorrelation Function
ANN	Artificial Neural Network
AIC	Akaike Information Criterion
ARIMA	Autoregressive Integrated Moving Average
BIC	Bayesian Information Criterion
DC	Datacenters
GHz	Gigahertz
IETF	Internet engineering Task Force
Mdt	Modelled time series
OS	Optical Spectrum
OXC	Optical Cross Connected
PACF	Partial Autocorrelation Function
VoD	Video on Demand

References

[Ad13] An Introductory Study On Time Series Modeling and Forecasting, Ratnadip Adhikari and R. K. Agrawal, LAP Lambert Academic Publishing (2013).

[Bre01] Breiman, Leo. Statistical Modeling: The Two Cultures. Statist. Sci. 16, no. 3, 199--231 (2001).

[Ca12] A. Castro, L.Velasco, M. Ruiz, M. Klinkowski, J. P. Fernández-Palacios, D. Careglio, "Dynamic routing and spectrum (re)allocation in future flexgrid optical networks", Computer Networks, vol. 56, pp 2869-2883 (2012).

[Co12] P. Cortez, M. Rio, M. Rocha and P. Sousa, "Multiscale Internet Traffic Forecasting using Neural Networks and Time Series Methods", in Expert Systems, Wiley-Blackwell, Vol.29 No.2. (May 2012)

[Di99] Dietterich, T. G. "Machine Learning", in Rob Wilson and Frank Keil (Eds.) The MIT Encyclopedia of the Cognitive Sciences, MIT Press. 497-498 (1999).

[Fa98] Julian Faraway and Chris Chatfiels, "Time series forecasting with neural networks: a comparative study using the airline data", Appl. Statist. 47, Part 2, pp. 231-250 (1998).

[Gi16] Ll. Gifre, L. M. Contreras, V. Lopez, and L. Velasco, "Big Data Analytics in Support of Virtual Network Topology Adaptability," in Proc. IEEE/OSA Optical Fiber Communication Conference (OFC), (2016).

[Gu03] Guyon, I. and Elisseeff, A., "An introduction to variable and feature selection", The Journal of Machine Learning Research 3: 1157–1182 (2003).

[Ha09] David J. Hand, "Mining the past to determine the future: Problems and possibilities", International Journal of Forecasting 25, 441-451 (2009).

[Ka95] R. Kass and A. Raftery, "Bayes Factors," Journal of the American

Statistical Association, vol. 90, pp. 773–795 (1995).

[Kl09] Kļevecka, I. “Forecasting Network Traffic: A Comparison of Neural Networks and Linear Models”. In: Abstracts of the 9th International Conference “Reliability and Statistics in Transportation and Communication”, pp. 21-24 (October 2009).

[Lo14] Lopez, R., “Open NN: An Open Source Neural Networks C++ Library [software]”. Retrieved from www.cimne.com/flood (2014).

[Mo16] F. Morales et al., “Virtual Network Topology Reconfiguration based on Big Data Analytics for Traffic Prediction,” in Proc. IEEE/OSA Optical Fiber Communication Conference (OFC) (2016)

[Na16] Z. Nasralla, T. El-Gorashi, M. Musa, J. Elmirghani, “Routing Post-Disaster Traffic Floods in Optical Core Networks,” in Proc. International Conference on Optical Network Design and Modelling (ONDM) (2016).

[OM] OMNET++: http://www.omnetpp.org/

[RFC7491] D. King and A. Farrel, “A PCE-Based Architecture for Application-Based Network Operations,” IETF RFC7491 (2015).

[Ru16-1] M. Ruiz, M. Germán, L. M. Contreras, and L. Velasco, "Big Data-backed Video Distribution in the Telecom Cloud," Elsevier Computer Communications, vol. 84, pp. 1-11 (2016).

[Ru16-2] M. Ruiz, F. Fresi, A. P. Vela, G. Meloni, N. Sambo, F. Cugini, L. Poti, L. Velasco, and P. Castoldi, "Service-triggered failure identification/localization through monitoring of multiple parameters," accepted in European Conference on Optical Communication (ECOC) (2016).

[Ve15] L. Velasco, L.M. Contreras, G. Ferraris, A. Stavdas, F. Cugini, M. Wiegand, and J. P. Fernández-Palacios, “A Service-Oriented Hybrid Access Network and Cloud Architecture,” IEEE Communications Magazine, vol. 53, pp. 159-165 (2015).

[Ve16-1] L. Velasco, F. Morales, Ll. Gifre, A. Castro, O. González de Dios, and M. Ruiz, "On-demand Incremental Capacity Planning in Optical Transport Networks," IEEE/OSA Journal of Optical Communications and Networking (JOCN), vol. 8, pp. 11-22 (2016).

[Ve16-2] A. P. Vela, A. Via, F. Morales, M. Ruiz, and L. Velasco, "Traffic generation for telecom cloud -based simulation," accepted in IEEE International Conference on Transparent Optical Networks (ICTON) (2016).

[Ve16-3] A. P. Vela, A. Via, M. Ruiz, and L. Velasco, "Bringing Data Analytics to the Network Nodes," accepted in European Conference on Optical

Communication (ECOC) (2016).

[Vi13] Vili Petek, “Polynomial fitting in C++ (not using Boost)”, http://vilipetek.com